Intel® Arc™ A-Series Graphics and Intel Data Center GPU Flex Series

Open-Source Programmer's Reference Manual

For the discrete GPUs code named "Alchemist" and "Arctic Sound-M"

Vol 2a: Command Reference: Instructions

March 2023, Revision 1.0

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks

Customer is responsible for safety of the overall system, including compliance with applicable safety-related requirements or standards.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exceptions that a) you may publish an unmodified copy and b) code included in this document is licensed subject to Zero-Clause BSD open source license (0BSD). You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

Table of Contents

3DMESH_1D

3DMESH_1D - 3DMESH_1D

Source: BSpec

Length Bias: 2

This command requests a 1D dispatch of (a) TaskShader ThreadGroups (TaskTGs) if TaskShaderEnabled, or (b) MeshShader ThreadGroups (MeshTGs) if TaskShaderDisabled.
The 1D count of TGs specified by ThreadGroup Count X. The ThreadGroup Count X parameter may contain a full 32-bit value. The Starting ThreadGroup ID X parameter is used to determine the starting value of the increasing 1D TGIDs.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 3h 3DMESH Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1h 3DMESH_1D Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14	**Extended Parameter 0 Present** Format: Boolean If this bit set, the Extended Parameter 0 (XP0) DWord is included in this command. If this bit set, 3DSTATE_TASK_SHADER::XP0Required is set and TaskShaderEnabled, the XP0 value passed will be included in TaskShader payloadsas XP0. If this bit setand 3DSTATE_MESH_SHADER::XP0Required is set, the XP0 value passed will be included in MeshShader payloads as XP0. If this bit is clear and 3DSTATE_TASK_SHADER::XP0Required is set and TaskShaderEnabled, zero will be included as the XP0 value in the TaskShader payload.. Likewise, if 3DSTATE_MESH_SHADER::XP0Required is set, zero will be included as the XP0 value in the MeshShader payloads.
	13	**TBIMR Enabled** This bit represents whether or not the driver wants to include this 3DMESH's objects to be tiled post setup. This bit is passed as part of the pipeline state.

<table>
<tr><th colspan="3">3DMESH_1D - 3DMESH_1D</th></tr>
<tr><td></td><td>12:11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Indirect Parameter Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>If set, the values in DW1 and beyondare ignored and replaced by the current values of specific MMIO registers, Refer to the parameter descriptions below for which MMIO register supplies a particular parameter value.</td></tr>
<tr><td></td><td>9</td><td><b>UAV Coherency Required</b><table><tr><td>Format:</td><td>U1</td></tr></table>SW will be required to set this bit if there is the possibility of sharing a UAV from a previous 3DMESH command. If set, this command may cause a flush due to UAV coherency requirements. If none of the shaders have UAV access enabled, then this bit is ignored.</td></tr>
<tr><td></td><td>8</td><td><b>Predicate Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>If set, this command is executed (or not) depending on the current value of the MI Predicate internal state bit. This command is ignored only if PredicateEnable is set and the Predicate state bit is 0.</td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>Total Length - 2. Excludes DWord (0,1).<table><tr><th>Value</th><th>Name</th><th>Exists If</th></tr><tr><td>1</td><td>Extended Parameter Not Present <b>[Default]</b></td><td>[Extended Parameter 0 Present] == FALSE</td></tr><tr><td>2</td><td>Extended Parameter Present</td><td>[Extended Parameter 0 Present] == TRUE</td></tr></table></td></tr>
<tr><td>1</td><td>31:0</td><td><b>ThreadGroup Count X</b><table><tr><td>Format:</td><td>U32</td></tr></table>This field specifies how many TaskShader or MeshShader threadgroups are to be dispatched in the X dimension.<table><tr><th>Programming Notes</th></tr><tr><td><ul><li>If Indirect Parameter Enable is set, this field is ignored and the ThreadGroup Count X is provided by the contents of the 3DMESH_TG_COUNT_X MMIO register.</li><li>Specifying a ThreadGroup Count Xvalue of 0 (directly or indirectly) effectively makes the command a 'no-operation'.</li><li>The values passed for Starting ThreadGroup ID X and ThreadGroup Count X shall not cause TGIDs to exceed (2^32)-1.</li></ul></td></tr></table></td></tr>
<tr><td>2</td><td>31:0</td><td><b>Starting ThreadGroup ID X</b><table><tr><td>Format:</td><td>U32</td></tr></table>This field specifies the ThreadGroup ID X (TGID.X) associated with the first threadgroup dispatched. TGIDs of subsequent threadgroups monotonically increase by 1.</td></tr>
</table>

<table>
<tr><th colspan="3">3DMESH_1D - 3DMESH_1D</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>

- If Indirect Parameter Enable is set, this field is ignored and the Starting ThreadGroup ID is provided by the contents of the 3DMESH_STARTING_TGID MMIO register.
- The values passed for Starting ThreadGroup ID X and ThreadGroup Count X shall not cause TGIDs to exceed (2^32)-1.

</td></tr>
</table>
</td></tr>
<tr><td>3</td><td>31:0</td><td>
<b>Extended Parameter 0 (XP0)</b>
<table>
<tr><td>Exists If:</td><td>[Extended Parameter 0 Present] == TRUE</td></tr>
<tr><td>Format:</td><td>U32</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>

- If Indirect Parameter Enable is set, this field is ignored and the Extended Parameter 0 is provided by the contents of the 3DPRIM_XP0 MMIO register.

</td></tr>
</table>
</td></tr>
</table>

3DMESH_3D

3DMESH_3D - 3DMESH_3D

Source: BSpec

Length Bias: 2

A 3D ThreadGroup dispatch of (a) TaskShader ThreadGroups (TaskTGs) if TaskShaderEnabled, or (b) MeshShader ThreadGroups (MeshTGs) if TaskShaderEnabled,is requested.
ThreadGroup Count X, Y and Z parameters are included in this command. The number of ThreadGroups dispatched is specified by the product of all three ThreadGroup Count parameters and is limited to 2^22.
ThreadGroup Count fields are each limited to 16 bits, with the upper bits MBZ.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 3h 3DMESH Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 2h 3DMESH_3D Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14	**Extended Parameter 0 Present** Format: Boolean If this bit set, the Extended Parameter 0 (XP0) DWord is included in this command. If this bit set, 3DSTATE_TASK_SHADER::XP0Required is set and TaskShaderEnabled, the XP0 value passed will be included in TaskShader payloadsas XP0. If this bit setand 3DSTATE_MESH_SHADER::XP0Required is set, the XP0 value passed will be included in MeshShader payloads as XP0. If this bit is clear and 3DSTATE_TASK_SHADER::XP0Required is set and TaskShaderEnabled, zero will be included as the XP0 value in the TaskShader payload.. Likewise, if 3DSTATE_MESH_SHADER::XP0Required is set, zero will be included as the XP0 value in the MeshShader payloads. If 3DSTATE_MESH_SHADER::XP0Required is clear, the XP0 parameter (if present) will be ignored and the XP0 value included in payloads will be UNDEFINED.

<table>
<tr><th colspan="3">3DMESH_3D - 3DMESH_3D</th></tr>
<tr><td></td><td>13</td><td><b>TBIMR Enabled</b>
This bit represents whether or not the driver wants to include this 3DMESH's objects to be tiled post setup. This bit is passed as part of the pipeline state.</td></tr>
<tr><td></td><td>12:11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Indirect Parameter Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>If set, the values in DW1 and beyondare ignored and replaced by the current values of specific MMIO registers, Refer to the parameter descriptions below for which MMIO register supplies a particular parameter value.</td></tr>
<tr><td></td><td>9</td><td><b>UAV Coherency Required</b><table><tr><td>Format:</td><td>U1</td></tr></table>SW will be required to set this bit if there is the possibility of sharing a UAV from a previous 3DMESH command. If set, this command may cause a flush due to UAV coherency requirements. If none of the shaders have UAV access enabled, then this bit is ignored.</td></tr>
<tr><td></td><td>8</td><td><b>Predicate Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>If set, this command is executed (or not) depending on the current value of the MI Predicate internal state bit. This command is ignored only if PredicateEnable is set and the Predicate state bit is 0.</td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>Total Length - 2. Excludes DWord (0,1).<table><tr><th>Value</th><th>Name</th><th>Exists If</th></tr><tr><td>2</td><td>Extended Parameter Not Present <b>[Default]</b></td><td>[Extended Parameter 0 Present] == FALSE</td></tr><tr><td>3</td><td>Extended Parameter Present</td><td>[Extended Parameter 0 Present] == TRUE</td></tr></table></td></tr>
<tr><td>1</td><td>31:0</td><td><b>ThreadGroup Count X</b><table><tr><td>Format:</td><td>U32</td></tr></table><table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,65535]</td><td></td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td><ul><li>If Indirect Parameter Enable is set, this field is ignored and the ThreadGroup Count Xis provided by the contents of the 3DPRIM_TG_COUNT_XMMIO register.</li><li>Specifying a ThreadGroup Count Xvalue of 0 (directly or indirectly) effectively makes the command a 'no-operation'</li></ul></td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">3DMESH_3D - 3DMESH_3D</th></tr>
<tr><td>2</td><td>31:0</td><td><b>ThreadGroup Count Y</b>
<table><tr><td>Format:</td><td>U32</td></tr></table>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,65535]</td><td></td></tr></table>
<table><tr><th>Programming Notes</th></tr><tr><td><ul><li>If Indirect Parameter Enable is set, this field is ignored and the ThreadGroup Count Y is provided by the contents of the 3DPRIM_XP1 MMIO register.</li><li>Specifying a ThreadGroup Count Yvalue of 0 (directly or indirectly) effectively makes the command a 'no-operation'</li></ul></td></tr></table>
</td></tr>
<tr><td>3</td><td>31:0</td><td><b>ThreadGroup Count Z</b>
<table><tr><td>Format:</td><td>U32</td></tr></table>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,65535]</td><td></td></tr></table>
<table><tr><th>Programming Notes</th></tr><tr><td><ul><li>If Indirect Parameter Enable is set, this field is ignored and the ThreadGroup Count Z is provided by the contents of the 3DPRIM_XP2 MMIO register.</li><li>Specifying a ThreadGroup Count Z value of 0 (directly or indirectly) effectively makes the command a 'no-operation'.</li></ul></td></tr></table>
</td></tr>
<tr><td>4</td><td>31:0</td><td><b>Extended Parameter 0 (XP0)</b>
<table><tr><td>Exists If:</td><td>[Extended Parameter 0 Present] == TRUE</td></tr><tr><td>Format:</td><td>U32</td></tr></table>
<table><tr><th>Programming Notes</th></tr><tr><td><ul><li>If Indirect Parameter Enable is set, this field is ignored and the Extended Parameter 0 value is provided by the contents of the 3DPRIM_XP0 MMIO register.</li></ul></td></tr></table>
</td></tr>
</table>

3DPRIMITIVE

3DPRIMITIVE

Source:	RenderCS
Length Bias:	2

The 3DPRIMITIVE command is used to submit 3D primitives to be processed by the 3D pipeline. Typically the processing results in rendering pixel data into the render targets, but this is not required.

The parameters passed in this command are forwarded to the Vertex Fetch function. The Vertex Fetch function will use this information to generate vertex data structures and store them in the URB. These vertices are then passed down the 3D pipeline.

Programming Notes
If the threads spawned by this command are required to observe memory writes performed by threads spawned from a previous command, software must precede this command with a command that performs a (preferably pipelined) memory flush (e.g., 3D_PIPECONTROL).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 3h 3DPRIMITIVE Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0h 3DPRIMITIVE Format: OpCode
	15	**Reserved**
	14	**Reserved** Access: RO Format: MBZ
	13	**TBIMR Enabled** This bit represents whether or not the driver wants to include this 3DPRIMITIVE objects to be tiled post setup. This bit is passed as part of the pipeline state.
	12	**Reserved** Access: RO Format: MBZ

3DPRIMITIVE

	11	**Extended Parameters Present** Format: Boolean If true, three additional DWords containing XP0, XP1 and XP2 parameters are included in this command. Depending on the setting of Indirect Parameter Enable, XP0-2 parameters (sourced either from the inline XP0-2 DWords or indirectly from the 3DPRIM_XP0-2 registers) are passed to the VF stage as possible sources for VF SGV insertion. If false, XP0-2 DWords are not included in this command and instead XP0-2 values default to zero if enabled as sources in VF SGV insertion.
	10	**Indirect Parameter Enable** Format: Enable If set, the values in DW 2-5 are ignored and replaced by the current values of the corresponding 3DPRIM_xxx MMIO registers: • 3DPRIM_VERTEX_COUNT (instead of DW2: VertexCountPerInstance) • 3DPRIM_START_VERTEX (instead of DW3: StartVertexLocation) • 3DPRIM_INSTANCE_COUNT (instead of DW4: InstanceCount) • 3DPRIM_START_INSTANCE (instead of DW5: StartInstanceLocation) • 3DPRIM_BASE_VERTEX (instead of DW6: BaseVertexLocation) Indirect Parameter Enable and End Offset Enable shall not be ENABLED at the same time, or behavior is UNDEFINED. If set and Extended Parameters Present is true, the current contents of the 3DPRIM_XP0-2 MMIO registers are passed to the VF stage as possible sources for VF SGV insertion and the Extended Parameter 0-2 values included in this command are ignored.
	9	**UAV Coherency Required** Format: U1 SW will be required to set this bit if there is the possibility of sharing a UAV from a previous 3DPRIMITVE command. If set, this command may cause a flush due to UAV coherency requirements. If none of the shaders have UAV access enabled, then this bit is ignored.
	8	**Predicate Enable** Format: Enable If set, this command is executed (or not) depending on the current value of the MI Predicate internal state bit. This command is ignored only if PredicateEnable is set and the Predicate state bit is 0.

<table>
<tr><th colspan="3">3DPRIMITIVE</th></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Format:</td><td>=n</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Programming Notes</th><th>Exists If</th></tr>
<tr><td>5h</td><td>5</td><td></td><td>[Extended Parameter Enable] == 'false'</td></tr>
<tr><td>8h</td><td>8 <b>[Default]</b></td><td>When Extended Parameter Enable is true, three Dwords containing Extended Parameter 0-2 shall be included in this command and the DWord Length field set to account for these additional DWords.</td><td>[Extended Parameter Enable] == 'true'</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>31:10</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>9</td><td><b>End Offset Enable</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
<table>
<tr><td>If set, the Vertex Count Per Instance field is IGNORED, and the 3DPRIM_END_OFFSET register is used to indirectly specify the vertex count by defining the amount of valid data in VB0. The following restrictions apply:
• VB0 must be enabled for use
• VertexAccessType = SEQUENTIAL
• Start Vertex Location = 0
• Start Instance Location = 0
• Base Vertex Location = 0</td></tr>
<tr><td>Vertices are output until EndOffset is reached or exceeded in VB0. If EndOffset is reachedor exceeded within the data associated with a vertex, that vertex is considered incompleteand will not be output. Partial objects will be discarded (as is normally done).
If clear, End Offset is ignored.
Indirect Parameter Enable and End Offset Enable must not be ENABLED at the same time, or behavior is UNDEFINED.</td></tr>
</table>
</td></tr>
<tr><td></td><td>8</td><td><b>Vertex Access Type</b>
This field specifies how data held in vertex buffers marked as VERTEXDATA is accessed by Vertex Fetch.</td></tr>
</table>

3DPRIMITIVE

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>SEQUENTIAL</td><td>VERTEXDATA buffers are accessed sequentially Required if End Offset Enable is ENABLED.</td></tr>
<tr><td>1h</td><td>RANDOM</td><td>VERTEXDATA buffers are accessed randomly via an index obtained from the Index Buffer.</td></tr>
</table>
</td></tr>
<tr><td></td><td>7:6</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>5:0</td><td>
<b>Primitive Topology Type</b>
<table>
<tr><td>Format:</td><td>3D_Prim_Topo_Type</td></tr>
</table>
<table>
<tr><td>This field specifies the topology type of 3D primitive generated by this command. Note that a single primitive topology (list/strip/fan/etc.) can contain a number of basic objects (lines, triangles, etc.).</td></tr>
<tr><td>This field is ignored. The topology type is specified via the 3DSTATE_VF_TOPOLOGY command.</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>31:0</td><td>
<b>Vertex Count Per Instance</b>
<table>
<tr><td>Format:</td><td>U32</td></tr>
</table>
This field specifies how many vertices are to be generated for each instance of the primitive topology. If End Offset Enable is clear: Format = U32 count of vertices Range = [0, 2^32-1] (upper limit probably constrained by VB size)Ignored if End Offset Enable or Indirect Parameter Enable is ENABLED.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td><ul><li>This per-instance value should specify a valid number of vertices for the primitive topology type. E.g., for 3DPRIM_TRILIST_ADJ, this field should specify a multiple of 6 vertices. However, in cases where too few or too many vertices are provided, the unused vertices will be silently discarded by the pipeline.</li><li>A 0 value is this field effectively makes the command a 'no-operation'.</li></ul></td></tr>
</table>
</td></tr>
<tr><td>3</td><td>31:0</td><td>
<b>Start Vertex Location</b>
<table>
<tr><td>Format:</td><td>U32</td></tr>
</table>
This field specifies the "starting vertex" for each instance. This allows skipping over part of the vertices in a buffer if, for example, a previous 3DPRIMITIVE command had already drawn the primitives associated with the earlier entries. For SEQUENTIAL access, this field specifies, for each instance, a starting structure index into the vertex buffers For RANDOM access, this field specifies, for each instance, a starting index into the Index Buffer.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td><ul><li>Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0).</li></ul></td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th colspan="3">3DPRIMITIVE</th></tr>
<tr><td></td><td></td><td><ul><li>Must be set to 0 if End Offset Enable is ENABLED.</li><li>Ignored if Indirect Parameter Enable is ENABLED</li></ul></td></tr>
<tr><td>4</td><td>31:0</td><td><b>Instance Count</b><table><tr><td>Format:</td><td>U32</td></tr></table>This field specifies the number of instances by which the primitive topology is to be regenerated. A value of 0 indicates "no instances" (no-op operation). A value of 1 effectively specifies "non-instanced" operation, though vertex buffers will still be used to provide instance data, if so programmed. Ignored if Indirect Parameter Enable is ENABLED.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0, FFFFFFFFh]</td><td></td></tr></table></td></tr>
<tr><td>5</td><td>31:0</td><td><b>Start Instance Location</b><table><tr><td>Format:</td><td>U32</td></tr></table><table><tr><td>This field specifies the "starting instance" for the command as an initial structure index into Vertex Buffers for vertex elements with Instancing Enable set.</td></tr><tr><td>Subsequent instances will access sequential instance data structures, as controlled by the Instance Data Step Rate.</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td><ul><li>Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0).</li><li>Must be set to 0 if End Offset Enable is ENABLED.</li><li>Ignored if Indirect Parameter Enable is ENABLED.</li></ul></td></tr></table></td></tr>
<tr><td>6</td><td>31:0</td><td><b>Base Vertex Location</b><table><tr><td>Format:</td><td>S31</td></tr></table>This field specifies a signed bias to be added to values read from the index buffer. This allows the same index buffer values to access different vertex data for different commands. This field applies only to RANDOM access mode. This field is ignored for SEQUENTIAL access mode, where there Start Vertex Location can be used to specify different regions in the vertex buffers.<table><tr><th>Programming Notes</th></tr><tr><td><ul><li>Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0).</li><li>Must be set to 0 if End Offset Enable is ENABLED.</li><li>Ignored if Indirect Parameter Enable is ENABLED.</li></ul></td></tr></table></td></tr>
</table>

3DPRIMITIVE		
7 **Exists if:** [Extended Parameters Present] == TRUE	31:0	**Extended Parameter 0** Format: U32 If Indirect Parameter Enable is not set, this field specifies a U32 XP0 parameter value to be passed to the VF stage as a possible source for SGV insertion. If Indirect Parameter Enable is set, this field is ignored.
8 **Exists if:** [Extended Parameters Present] == TRUE	31:0	**Extended Parameter 1** Format: U32 If Indirect Parameter Enable is not set, this field specifies a U32 XP1 parameter value to be passed to the VF stage as a possible source for SGV insertion. If Indirect Parameter Enable is set, this field is ignored.
9 **Exists if:** [Extended Parameters Present] == TRUE	31:0	**Extended Parameter 2** Format: U32 If Indirect Parameter Enable is not set, this field specifies a U32 XP2 parameter value to be passed to the VF stage as a possible source for SGV insertion. If Indirect Parameter Enable is set, this field is ignored.

3DSTATE_3D_MODE

3DSTATE_3D_MODE

Source:	RenderCS
Length Bias:	2

This command is general 3D programming state that can be shared from the top to bottom of the pipeline.

Programming Notes
if SW needs to program any bitfield in bit group 2, it also has to program bitfield [0] of bit group 2 to 1 If SW needs to program any bitfield in bit group 4, it has to program it two times. The first time, it has to set bit field[0] of bit group 4 to 1. The second time, it has to do the same programming with bit field[0] of bit group 4 set to 0.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h GFXPIPE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Eh 3DSTATE_3D_MODE Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 3h Excludes DWord (0,1) Format: =n
1	31:16	**Mask Bits 1** Format: Enable[16] This field is the mask bits for the state bits below. This is a bit wise mask where the bit number-16 is the value of the corresponding bit being masked in the same data word. For example, if you want to update state for bits 3:2 then bits 19:18 must be set.
	15:11	**Reserved** Access: RO Format: MBZ

3DSTATE_3D_MODE

10 **Reserved**

Access:	RO
Format:	MBZ

9 **Float Blend Optimization Disable**

Value	Name	Description
0h	**[Default]**	Enables blend optimization for floating point RTs.
1h		Disables blend optimization for floating point RTs.

8 **Reserved**

Access:	RO
Format:	MBZ

7 **Reserved**

Access:	RO
Format:	MBZ

6 **Slice Hashing Table Enable**

Format:	Enable

This field enables the use of indirect state **SLICE_HASH_TABLE** programmed via **3DSTATE_SLICE_HASH_STATE_POINTER**.

Value	Name	Description
0h	Disable **[Default]**	Slice Hashing is based on default lookup tables with the following function (X+Y) % total_enabled_subslices % enabled_slices
1h	Enable	Slice Hashing is via SLICE_HASH_TABLE from 3DSTATE_SLICE_HASH_STATE_POINTER[total_enabled_subslices-4]

5 **Subslice Hashing Table Enable**

Format:	Enable

This field enables the use of the Subslice Hashing Mode table programmed via **3DSTATE_SUBSLICE_HASH_TABLE**.

Value	Name	Description
0h	Disable **[Default]**	Subslice Hashing is computed
1h	Enable	Subslice Hashing is via 3DSTATE_SUBSLICE_HASH_TABLE

4 **3D Scoreboard Hashing Mode**

Value	Name	Description
0	Scoreboard address calculation optimized for Subslice Hashing Mode **[Default]**	Before scoreboard address calculations one address bit will be removed to increase the efficiency of the scoreboard.
1	Scoreboard address calculation not optimized	All address bits used when calculating scoreboard address.

3DSTATE_3D_MODE

Programming Notes
When **Subslice Hashing Table Enable** and table entries do not contain a checkerboard pattern for each subslice, then optimizing for **Subslice Hashing Mode** may result in reduced performance due to increased false dependencies.

3:2 **Subslice Hashing Mode**

This field is not used by the hardware. Hardware performs 16x16 hashing only.

1:0 **Cross Slice Hashing Mode**

Format:	U2

Value	Name	Description	Programming Notes
0h	Normal Mode **[Default]**	num_slices > 1 : 16x16 Hashing enabled num_slices == 1: No cross slice hashing	
1h	Cross Slice Hashing Disable	Disables the cross slice hashing	
2h	Reserved	Reserved	
3h	32X32 hashing	32X32 pixel hashing across slices	This setting must be used when sub-slice hashing mode is 16x16 and num_slices > 1

Programming Notes
A stalling flush is required before changing the value of this field. This is to make sure the entire pipeline is drained.

2 31:16 **Mask Bits 2**

Format:	Enable[16]

This field is the mask bits for the state bits below. This is a bit wise mask where the bit number-16 is the value of the corresponding bit being masked in the same data word. For example, if you want to update state for bits 6:0 then bits 22:16 must be set.

15:10 **Reserved**

Access:	RO
Format:	MBZ

9 **Reserved**

Access:	RO
Format:	MBZ

8 **HDC NULL PAGE Disable**

Access:	R/W
Format:	Disable
_Custom_GTIReset:	DEV

By default, enable detection of NULL pages that return zero on reads and drop writes.

3DSTATE_3D_MODE

Value	Name	Description
0h	Enable **[Default]**	When Reset, HDC supports the Null Page feature.
1h	Disable	When Set, HDC does not support the Null Page feature.

7 **SRC Clear WO Allocation in RCC**

Access:	R/W
Format:	Disable
_Custom_GTIReset:	DEV

By default, enable WO allocation for src clear.

Value	Name	Description
0h	Enable **[Default]**	When Reset, SRC clear is allocated as WO in RCCunit
1h	Disable	When Set, SRC clear is allocated as RW in RCCunit

6:0 **AMFS MOCS**

Format:	MEMORY_OBJECT_CONTROL_STATE

Programming Notes
SW must explicitly program AMFS_FLUSH using PIPE_CONTROL command prior to changing this field. SW must also explicitly flush DC cache flush if the setting of this field is changing from cacheable to non-cacheable in L3.

3: 31:16 **Mask Bits 3**

Format:	Enable[16]

This field is the mask bits for the state bits below. This is a bit wise mask where the bit number-16 is the value of the corresponding bit being masked in the same data word. For example, if you want to update state for bits 6:0 then bits 22:16 must be set.

15 **RCC RHWO Optimization Disable**

Access:	R/W
_Custom_GTIReset:	DEV

Value	Name
0	Enable
1h	Disable **[Default]**

14:12 **MSAA Compression Plane Number Threshold for eLLC**

Access:	R/W
_Custom_GTIReset:	DEV

Value	Name	Description
0h	threshold0 **[Default]**	Cache only planeID = 0 in eLLC.
1h	threshold1	Cache only planeID = 0, 1 in eLLC.
2h	threshold2	Cache only planeID = 0..2 in eLLC.

3DSTATE_3D_MODE

3h	threshold3	Cache only planeID = 0..3 in eLLC.
4h	threshold4	Cache only planeID = 0..4 in eLLC.
5h	threshold5	Cache only planeID = 0..5 in eLLC.
6h	threshold6	Cache only planeID = 0..6 in eLLC.
7h	threshold7	Cache only planeID = 0..7 in eLLC.

Programming Notes
This bit-field is programmed based on MSAA. When MSAA compression is enabled, these settings affect HW, else it is ignored. For 16X MSAA only lower 8 planes can be cached in eLLC.

11 **RCPB RAW stall optimization disable**

Access:	R/W
_Custom_GTIReset:	DEV

Value	Name	Description
0h	Enable **[Default]**	Enables RCPB RAW stall optimization
1h	Disable	Disables RCPB RAW stall optimization

10 **Disable 64bpp TileY perf fix**

Value	Name	Description
0	**[Default]**	Enable 64bpp TiledY perf fix for 2x/8x SIMD8
1		Disable 64bpp TiledY perf fix (see description in HSD)

10 **Fast Clear Optimization (FCV) Enable**

Value	Name	Description
0	Disable **[Default]**	When set to 0, fast clear optimization is disabled in RCC
1	Enable	When set to 1, fast clear optimization is enabled in RCC

9 **RCC set mapping mode**

Value	Name	Description
1	Disable	select legacy set mapping
0	Enable **[Default]**	Selects new set mapping which is cognizant of L3 bank hashing and 512B node hashing

8 **Reserved**

7 **Disable RCC Dirty-bit Based Eviction Policy**

Access:	R/W
_Custom_GTIReset:	DEV

If set, disables the dirty-bit based eviction policy bit only keeps first-available since LRA eviction policy. If reset, enables dirty-bit based eviction policy along with first available since LRA eviction policy.

3DSTATE_3D_MODE

Value	Name	Description
0h	Enable **[Default]**	Enables the dirty-based eviction policy
1h	Disable	Disables the dirty-based eviction policy

6 **PTBR discard in L1 disable**

Disable PTBR discard in L1 (MSC and RCC)

Value	Name	Description
0	**[Default]**	Allow PTBR discard in L1 caches in Pixel Pipe
1		Disable PTBR discards from L1 caches in Pixel Pipe

5 **MCS Cache Disable**

Access:	R/W
Format:	Disable
_Custom_GTIReset:	DEV

For Programming restrictions please refer to the 3D Pipeline.

Value	Name	Description
0h	**[Default]**	MCS cache enabled. It allows RTs with MCS buffer enabled to be rendered using either MSAA compression for MSRT OR with color clear feature for non MSRT.
1h		MCS cache is disabled. Hence no MSAA compression for MSRT and no color clear for non-MSRT.

4 **RCC Eviction Policy**

Access:	R/W
Format:	Disable
_Custom_GTIReset:	DEV

If this bit is set, RCC unit will have LRA as replacement policy. The default value i.e.(when this bit is reset) indicates that non-LRA eviction policy. This bit must be reset. LRA replacement policy is not supported.

Programming Notes
If this bit is set to "1", bit 7 of 0x7010h must also be set to "1".

3 **RCC discard state machine optimization Disable**

Value	Name	Description
0	**[Default]**	Enable RCC discard state machine optimization (out of order set completion)
1		Disable RCC discard state machine optimization (out of order set completion)

3DSTATE_3D_MODE

2

RHWO 16 Max Outstanding Request From RCC to CC

Access:	R/W
Format:	Disable
_Custom_GTIReset:	DEV

By default, max 16 outstanding requests are sent from RCC to CC.

Value	Name	Description
0h	Enable **[Default]**	When Reset, max 16 outstanding RHWO requests are sent from RCC to CC.
1h	Disable	When Set, max 30 outstanding RHWO requests are sent from RCC to CC.

1

Disable clock gating in the pixel backend

Access:	R/W
Format:	Disable
_Custom_GTIReset:	DEV

MCL related clock gating is disabled in the pixel backend. Before setting this bit to 1,the instruction/state caches must be invalidated.

0

Disable Byte sharing for 3D TYF LOD1 surfaces for 32/64/128 bpp

Access:	R/W
_Custom_GTIReset:	DEV

Value	Name	Description
1		Enable byte sharing for 3D TYF LOD1 surfaces - 32/64/128 bpp
0	**[Default]**	Disable Byte Sharing for 3D TYF surfaces LOD1 , 32/64/128 bpp

4 31:16

Mask Bits 4

Format:	Enable[16]

This field is the mask bits for the state bits below. This is a bit wise mask where the bit number-16 is the value of the corresponding bit being masked in the same data word. For example, if you want to update state for bits 3:2 then bits 19:18 must be set.

15

Disable Source Clear Cam Match fix in RCPBE

Access:	R/W
_Custom_GTIReset:	DEV

Value	Name	Description
0	**[Default]**	Enables the source clear cam match fix in PBE
1		Disables the source clear cam match fix in PBE

14:13

Reserved

Access:	RO
Format:	MBZ

3DSTATE_3D_MODE

12 **Lossless Compressed Cache Line Hash Selection**

Access:	R/W
Format:	Enable
_Custom_GTIReset:	DEV

Value	Name	Description
0h	Disabled **[Default]**	If this field is 0, address hash will not be based off Virtual Address bits (20:6). The address hash will only include bits (11:6)
1h	Enabled	If this field is 1, address hash will be based off Virtual Address bits (20:6). This is expected to avoid Hot Spot on eLLC/eDRAM channels and improve overall performance

Programming Notes
Hash method should be consistent with that of sampler(0xE194, bit 8) and display(0x42080, bit15)

11:9 **Reserved**

Access:	RO
Format:	MBZ

8 **Reserved**

7:6 **Encoding for fine grained performance throttling in RCPBE**

Access:	R/W
_Custom_GTIReset:	DEV

RCPBE will insert stalls to RCPBE, once every two cycles or three times every four cycles to throttle the pixel backend throughput in a fine-grained manner.

Value	Name	Description
0h	No stall **[Default]**	No stall
1h	Stall Alternate Cycles	Stall upstream unit every alternate cycle
2h	Stall Three of Four Cycles	Stall upstream unit three out of every four cycles
3h	Reserved	

5 **Disable Pixel Mask Based Camming in RCPBE**

Access:	R/W
Format:	Disable
_Custom_GTIReset:	DEV

By default, pixel mask-based camming in RCPBE unit is enabled.

Value	Name	Description
0h	Enable **[Default]**	When Reset, pixel mask based camming in RCPBE unit is enabled.
1h	Disable	When Set, pixel mask based camming in RCPBE unit is disabled.

3DSTATE_3D_MODE

4 **Disable Cam Reset Fix RCPBE**

Access:	R/W
Format:	Disable
_Custom_GTIReset:	DEV

By default, cam reset fix in RCPBE is enabled.

Value	Name	Description
0h	Enable **[Default]**	When Reset, cam reset fix in RCPBE is enabled.
1h	Disable	When Set, cam reset fix in RCPBE is disabled.

3 **Reserved**

Access:	RO
Format:	MBZ

2 **Tiled-Resource Mip Tail Layout for Volumetric Surfaces Disable**

Access:	R/W
_Custom_GTIReset:	DEV

When set, forces Mip Tail Layout for volumetric surfaces. When cleared, forces Mip Tail Layout for volumetric surfaces. Ignored for non-volumetric surfaces.

Value	Name	Description
0h	Disable **[Default]**	Forces Mip Tail Layout for volumetric surfaces.
1h	Enable	Forces Mip Tail Layout for volumetric surfaces.

1:0 **URB Hash Mode**

This field specifies the hash mode for the local URB. Modifying this value will change the hash algorithm and possibly the bank distribution of reads and writes.

Value	Name	Description
0h	Hashed bits 6 to 8 **[Default]**	Bit 6 Hash =A[6] ^ A[7] ^ A[10] ^ A[11] ^ A[14] ^ A[15] ^ A[18] Bit 7 Hash =A[7] ^ A[8] ^ A[11] ^ A[12] Bit 8 Hash=A[8] ^ A[9] ^ A[13] ^ A[14] Address Hash[18:6] = {Address[18:9], Bit 8 Hash, Bit 7Hash, Bit 6Hash} Node = Address Hash[6] Bank = Address Hash[18:7] %3 Bank Address = Address Hash[18:7] / 3

1h	URB Legacy Hash Mode	Bit 6 Hash =A[6] ^ A[8] ^ A[10] ^ A[12] ^ A[14] ^ A[16] ^ A[18]
		Address Hash[18:6] = {Address[18:7], Bit 6Hash}
		Node = Address Hash[6]
		Bank = Address Hash[18:7] %3
		Bank Address = Address Hash[18:7] / 3

2h,3h	Reserved	

3DSTATE_AA_LINE_PARAMETERS

3DSTATE_AA_LINE_PARAMETERS

Source:	RenderCS
Length Bias:	2

The 3DSTATE_AA_LINE_PARAMS command is used to specify the slope and bias terms used in the improved alpha coverage computation (specifically for DX WHQL compliance). Note that in these devices the coverage values passed to PS threads are full U0.8 values, versus where U0.4 values are passed.

Workaround
Workaround : This command must be followed by a PIPE_CONTROL with CS Stall bit set.,

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Ah 3DSTATE_AA_LINE_PARAMETERS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**Dword Length** Default Value: 1h Excludes Dword (0,1) Format: =n
1	31:24	**AA Point Coverage Bias** Format: U0.8 This field specifies the bias term to be used in the aa coverage computation for edges 0 and 3.
	23:16	**AA Coverage Bias** Format: U0.8 This field specifies the bias term to be used in the aa coverage computation for edges 0 and 3.

3DSTATE_AA_LINE_PARAMETERS

	15:8	**AA Point Coverage Slope** Format: | U0.8 This field specifies the slope term to be used in the aa coverage computation for edges 0 and 3.If this field is zero, the Windower will revert to legacy aa line coverage computation (though still output expanded U0.8 coverage values).
	7:0	**AA Coverage Slope** Format: | U0.8 This field specifies the slope term to be used in the aa coverage computation for edges 0 and 3.If this field is zero, the Windower will revert to legacy aa line coverage computation (though still output expanded U0.8 coverage values).
2	31:24	**AA Point Coverage EndCap Bias** Format: | U0.8 This field specifies the bias term to be used in the aa coverage computation for edges 1 and 2.
	23:16	**AA Coverage EndCap Bias** Format: | U0.8 This field specifies the bias term to be used in the aa coverage computation for edges 1 and 2.
	15:8	**AA Point Coverage EndCap Slope** Format: | U0.8 This field specifies the slope term to be used in the aa coverage computation for edges 1 and 2.
	7:0	**AA Coverage EndCap Slope** Format: | U0.8 This field specifies the slope term to be used in the aa coverage computation for edges 1 and 2.

3DSTATE_BINDING_TABLE_POINTERS_DS

3DSTATE_BINDING_TABLE_POINTERS_DS

Source:	RenderCS
Length Bias:	2

The 3DSTATE_BINDING_TABLE_POINTERS_DS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 28h 3DSTATE_BINDING_TABLE_POINTERS_DS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Binding Table Pointers State Body** Format: 3DSTATE_BINDING_TABLE_POINTERS_BODY

3DSTATE_BINDING_TABLE_POINTERS_GS

3DSTATE_BINDING_TABLE_POINTERS_GS		
Source: RenderCS Length Bias: 2		
The 3DSTATE_BINDING_TABLE_POINTERS_GS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 29h 3DSTATE_BINDING_TABLE_POINTERS_GS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Binding Table Pointers State Body** Format: 3DSTATE_BINDING_TABLE_POINTERS_BODY

3DSTATE_BINDING_TABLE_POINTERS_HS

3DSTATE_BINDING_TABLE_POINTERS_HS	
Source:	RenderCS
Length Bias:	2

The 3DSTATE_BINDING_TABLE_POINTERS_HS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 27h 3DSTATE_BINDING_TABLE_POINTERS_HS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Binding Table Pointers State Body** Format: 3DSTATE_BINDING_TABLE_POINTERS_BODY

3DSTATE_BINDING_TABLE_POINTERS_PS

3DSTATE_BINDING_TABLE_POINTERS_PS

Source:	RenderCS
Length Bias:	2

The 3DSTATE_BINDING_TABLE_POINTERS_PS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 2Ah 3DSTATE_BINDING_TABLE_POINTERS_PS Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Binding Table Pointers State Body** Format: 3DSTATE_BINDING_TABLE_POINTERS_BODY

3DSTATE_BINDING_TABLE_POINTERS_VS

3DSTATE_BINDING_TABLE_POINTERS_VS

Source:	RenderCS
Length Bias:	2

The 3DSTATE_BINDING_TABLE_POINTERS_VS command is used to define the location of fixed functions' BINDING_TABLE_STATE. Only some of the fixed functions utilize binding tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 26h 3DSTATE_BINDING_TABLE_POINTERS_VS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Binding Table Pointers State Body** Format: 3DSTATE_BINDING_TABLE_POINTERS_BODY

3DSTATE_BINDING_TABLE_POOL_ALLOC

3DSTATE_BINDING_TABLE_POOL_ALLOC

Source:	RenderCS, ComputeCS
Length Bias:	2

This command is to program the base address and size of the binding table pool. The address to fetch the binding table is based on the Binding Table Pool Base Address and the binding table pointer if the Binding Table Pool is enabled. Otherwise the binding table pointer is an offset from the Surface Base Address.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 19h 3DSTATE_BINDING_TABLE_POOL_ALLOC Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 2h DWORD_COUNT_n Format: =n
1..2	63:12	**Binding Table Pool Base Address** Format: VIRTUAL_ADDR[63:12] Specifies the 4K-byte aligned base address for Binding Table Pool. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	11	**Reserved** Access: RO Format: MBZ
	10:7	**Reserved** Access: RO Format: MBZ

3DSTATE_BINDING_TABLE_POOL_ALLOC		
	6:0	**Surface Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this surface. **Programming Notes** Bit 0 is not programmable and is always zero.
3	31:12	**Binding Table Pool Buffer Size** Format: U20 This field specifies the size of the buffer in 4K pages. Any access which straddle or go past the end of the buffer will return 0. Value | Name | Description [0,1048575] | | 0 | No Valid Data | There is no valid data in the buffer **Restriction** Programming size of zero is illegal in the case that the pool is enabled.
	11:0	**Reserved** Access: RO Format: MBZ

3DSTATE_BLEND_STATE_POINTERS

3DSTATE_BLEND_STATE_POINTERS

Source:	RenderCS
Length Bias:	2

The 3DSTATE_BLEND_STATE_POINTERS command is used to set up the pointers to the color calculator state.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 24h 3DSTATE_BLEND_STATE_POINTERS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Blend State Pointer State Body** Format: **3DSTATE_BLEND_STATE_POINTERS_BODY**

3DSTATE_BTD_CONSTANT_POINTER

3DSTATE_BTD_CONSTANT_POINTER

Source:	CommandStreamer
Length Bias:	2

This command sets pointers to the Global Arguments for Bindless Thread Dispatch. The constant data pointed to by this command is stored in memory, not in push constant buffer (PCB).
The 3DSTATE_BTD_CONSTANT_POINTER command gets committed to the shader on parsing 3DPRIMTIVE command or on encountering an explicit/implicit flush.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 73h 3DSTATE_CONSTANT_TS_POINTER Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Constant Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for constant buffers defined in this command.
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n n = Total Length -2
1..2	63:0	**Constant TS Pointer State Body** Format: 3DSTATE_BTD_CONSTANT_POINTER_BODY

3DSTATE_BTD

<table>
<tr><th colspan="3">3DSTATE_BTD</th></tr>
<tr><td colspan="3">Source: BSpec
Length Bias: 1</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h GFXPIPE</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Command SubType</b><table><tr><td>Default Value:</td><td>0h GFXPIPE_COMMON</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><b>3D Command Opcode</b><table><tr><td>Default Value:</td><td>1h GFXPIPE_NONPIPELINED</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>3D Command Sub Opcode</b><table><tr><td>Default Value:</td><td>06h 3DSTATE_BTD</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>15:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>n = Total Length -2<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>04h</td><td>DWORD_COUNT_n <b>[Default]</b></td><td>Excludes DWord (0,1)</td></tr></table></td></tr>
<tr><td>1..5</td><td>159:0</td><td><b>BTD State Body</b><table><tr><td>Format:</td><td>3DSTATE_BTD_BODY</td></tr></table></td></tr>
</table>

3DSTATE_CC_STATE_POINTERS

3DSTATE_CC_STATE_POINTERS		
Source:	RenderCS	
Length Bias:	2	
The 3DSTATE_CC_STATE_POINTERS command is used to set up the pointers to the color calculator state.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Eh 3DSTATE_CC_STATE_POINTERS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**CC State Pointers Body** Format: **3DSTATE_CC_STATE_POINTERS_BODY**

3DSTATE_CHROMA_KEY

3DSTATE_CHROMA_KEY	
Source:	RenderCS, ComputeCS
Length Bias:	2

The 3DSTATE_CHROMA_KEY instruction is used to program texture color/chroma-key key values. A table containing four set of values is supported. The ChromaKey Index sampler state variable is used to select which table entry is associated with the map. Texture chromakey functions are enabled and controlled via use of the ChromaKey Enable texture sampler state variable. Texture Color Key (keying on a paletted texture index) is not supported.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: Opcode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: Opcode
	23:16	**3D Command Sub Opcode** Default Value: 04h 3DSTATE_CHROMA_KEY Format: Opcode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:30	**ChromaKey Table Index** Format: U2 Selects which entry in the ChromaKey table is to be loaded Value: [0,3] / Name: / Description:

<table>
<tr><th colspan="3">3DSTATE_CHROMA_KEY</th></tr>
<tr><td></td><td></td><td><table><tr><td>[0,3]</td><td></td><td>The range of legal values for this field will depend on the value of Bit 9 (<b>ChromaKey Table For Compute Command Stream Disable</b>) in MMIO register E184h
If Bit9 is set to 1h, then the valid range of this field is 0,3.
If Bit9 is cleared to 0h, then the valid range of this field is 0,1</td></tr></table></td></tr>
<tr><td></td><td>29:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>2</td><td>31:0</td><td><b>ChromaKey Low Value</b>
This field specifies the "low" (minimum) value of the chroma key range. Texel samples are considered "matching the key" if each component of the texel falls within the (inclusive) chroma range. See ChromaKey High Value for further format, programming info.</td></tr>
<tr><td>3</td><td>31:0</td><td><b>ChromaKey High Value</b>
This field specifies the "high" (maximum) value of the chroma key range. Texel samples are considered "matching the key" if each component of the texel falls within the (inclusive) chroma range.<table>
<tr><th>Programming Notes</th></tr>
<tr><td>ChromaKey values are specified using 8-bit channels. When using surface formats with less than 8 bits per channel, the device will expand channels by replicating the required number of MSBs into the LSBs of each channel. Software must account for this conversion when it programs Chromakey Low/High Values (e.g., by performing the same replication).</td></tr>
<tr><td>For channels that do not exist in the actual surface (e.g., Alpha channel for non-ARGB maps), software must explicitly program full range high/low values (High=FFh, Low=0h for formats using unsigned chroma key values, High=7Fh, Low=FFh for formats using sign magnitude chroma key values) in order to effectively remove the comparison of that field from the ChromaKey function.</td></tr>
<tr><td>For channels in SNORM format in the surface format, the value in the high/low value for that channel is interpreted in sign magnitude format. Negative zero value is not supported (use positive zero instead). For channels with mixed UNORM/SNORM formats (i.e. R5G5_SNORM_B6_UNORM), the ChromaKey is programmed as if all channels are SNORM.</td></tr>
<tr><td>YUV ChromaKey will use an interpolated chrominance value from the map for comparison to the chroma key values for those texels without chrominance due to downsampling. The chrominance value used is the average of values to the left and right of the texel in question.</td></tr>
<tr><td>It is UNDEFINED to program any component of the ChromaKey High Value to be less than the corresponding component of ChromaKey Low Value.</td></tr>
<tr><td>Format = interpreted according to associated texel format "class":</td></tr>
<tr><td>Only the surface formats listed as supported for chroma key in the surface formats table can be used with this feature. Use of any other surface format with chroma key enabled is UNDEFINED.</td></tr>
<tr><td><table>
<tr><th>Surface Format</th><th>31:24</th><th>23:15</th><th>16:8</th><th>7:0</th></tr>
<tr><td>ARGB and BC (DXT) formats</td><td>A</td><td>R</td><td>G</td><td>B</td></tr>
<tr><td>YCrCb formats</td><td>A</td><td>Cr</td><td>Y</td><td>Cb</td></tr>
</table></td></tr>
</table></td></tr>
</table>

3DSTATE_CLEAR_PARAMS

3DSTATE_CLEAR_PARAMS

Source:	RenderCS
Length Bias:	2

This command defines the depth clear value delivered as a pipelined state command. However, the state change pipelining isn't completely transparent (see restriction below).

HW will internally manage the draining pipe and flushing of the caches when this command is issued. The PIPE_CONTROL restrictions are removed.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 04h 3DSTATE_CLEAR_PARAMS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**Dword Length** Default Value: 1h Excludes Dword (0,1) Format: =n
1..2	63:0	**Clear Params State Body** Format: 3DSTATE_CLEAR_PARAMS_BODY

3DSTATE_CLIP_MESH

3DSTATE_CLIP_MESH

Source:	RenderCS
Length Bias:	2

The 3DSTATE_CLIP_MESH command is used to provide the Clipper with states required to process geometry generated by the Mesh Shader stage.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 81h 3DSTATE_CLIP_MESH Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Clip Mesh State Body** Format: 3DSTATE_CLIP_MESH_BODY

3DSTATE_CLIP

3DSTATE_CLIP

Source:	RenderCS
Length Bias:	2

Restriction
When expected in POCS command stream, this programs the state for CLR stage of the POCS pipeline

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 12h 3DSTATE_CLIP Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 02h Excludes DWord (0,1) Format: =n Total Length - 2
1..3	95:0	**Clip State Body** Format: 3DSTATE_CLIP_BODY

3DSTATE_CONSTANT_ALL

3DSTATE_CONSTANT_ALL

Source:	RenderCS, PositionCS
Length Bias:	2

This instruction species pointers and sizes of data to be fetched from memory and loaded as part of the shaders thread payload.

Programming Notes
The programming of enabled buffers is in ascending order where if buffer zero is programmed it will be the first pointer programmed and parsed. Any buffers being omitted must not be programmed.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 6Dh 3DSTATE_CONSTANT_ALL Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14:13	**Reserved** Access: RO Format: MBZ
	12:8	**Shader Mask** This bit specifies if the updated pointers are valid for specific shaders. Value / Name: xxxx1b Vertex Shader Update Enable xxx1xb Hull Shader Update Enable xx1xxb Domain Shader Update Enable x1xxxb Geometry Shader Update Enable 1xxxxb Pixel Shader Update Enable

3DSTATE_CONSTANT_ALL

	7:0	**DWord Length** Format: =n n = 2b (where b = # of pointers included) **Value** — **Name** 8 — DWORD_COUNT_n **[Default]** [0,8] — 0-4 Pointers
1	31	**Update Mode** If set, pointers that are not valid will retain their value. If clear, then all pointers not valid will be cleared with zero address and zero size. If pointer is valid, then the value of this field is a don't care and the value programmed is always loaded. **Value** — **Name** 0 — Clear pointer and size of non-valid pointers 1 — Retain value of non-valid pointers
	30:20	**Reserved** Access: RO Format: MBZ
	19:16	**Pointer Buffer Mask** This bit specifies which pointers are valid in this command. **Value** — **Name** xxx1b — Buffer 0 Valid xx1xb — Buffer 1 Valid x1xxb — Buffer 2 Valid 1xxxb — Buffer 3 Valid
	15:7	**Reserved** Access: RO Format: MBZ
	6:0	**Constant Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command.
2..n	255:0	**Constant All Data** Format: 3DSTATE_CONSTANT_ALL_DATA

3DSTATE_CONSTANT_DS

3DSTATE_CONSTANT_DS	
Source:	RenderCS
Length Bias:	2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Ah 3DSTATE_CONSTANT_DS Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Constant Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command.
	7:0	**DWord Length** Default Value: 9h Excludes DWord (0,1) Format: =n
1..10	319:0	**Constant Body** Format: 3DSTATE_CONSTANT(Body) See the 3DSTATE_CONSTANT(Body) format for the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS.

3DSTATE_CONSTANT_GS

3DSTATE_CONSTANT_GS

Source:	RenderCS
Length Bias:	2

This command sets pointers to the push constants for the GS unit. The constant data pointed to by this command will be loaded into the GS unit's push constant buffer (PCB).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 16h 3DSTATE_CONSTANT_GS Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Constant Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command.
	7:0	**DWord Length** Default Value: 9h Excludes DWord (0,1) Format: =n
1..10	319:0	**Constant Body** Format: 3DSTATE_CONSTANT(Body) Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS

3DSTATE_CONSTANT_HS

3DSTATE_CONSTANT_HS

Source:	RenderCS
Length Bias:	2

This command sets pointers to the push constants for the HS unit. The constant data pointed to by this command is loaded into the HS unit's push constant buffer (PCB).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 19h 3DSTATE_CONSTANT_HS Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Constant Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command.
	7:0	**DWord Length** Default Value: 9h Excludes DWord (0,1) Format: =n
1..10	319:0	**Constant Body** Format: 3DSTATE_CONSTANT(Body) Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS

3DSTATE_CONSTANT_PS

3DSTATE_CONSTANT_PS

Source:	RenderCS
Length Bias:	2

This command sets pointers to the push constants for the PS unit. The constant data pointed to by this command is loaded into the PS unit's push constant buffer (PCB).

Programming Notes
A 3DSTATE_GATHER_PS command must be dispatched along with any 3DSTATE_CONSTANT_PS command when the Gather Pool is enabled within a batch buffer.
The 3DSTATE_CONSTANT_* command is not committed to the shader unit until the corresponding (same shader) 3DSTATE_BINDING_TABLE_POINTER_* command is parsed. For example, the 3DSTATE_CONSTANT_VS command will not fetch the constant buffers from memory and make available to the shader until the 3DSTATE_BINDING_TABLE_POINTERS_VS is parsed by the render command streamer. In case of multiple 3DSTATE_CONSTANT_VS programmed prior to 3DSTATE_BINDING_TABLE_POINTER_VS, only the most recently programmed 3DSTATE_CONSTANT_VS will be committed. On usage model of enabling legacy mode is when Resource Streamer is not enabled.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 17h 3DSTATE_CONSTANT_PS Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Constant Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command.
	7:0	**Dword Length**

3DSTATE_CONSTANT_PS

		Default Value: 9h Excludes DWord (0,1) Format: =n
1..10	319:0	**Constant Body** Format: **3DSTATE_CONSTANT(Body)** Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS

3DSTATE_CONSTANT_VS

3DSTATE_CONSTANT_VS

Source:	RenderCS
Length Bias:	2

This command sets pointers to the push constants for VS unit. The constant data pointed to by this command is loaded into the VS unit's push constant buffer (PCB).

Workaround

Workaround:
The driver must ensure the following case does not occur without a flush to the 3D engine: 3DSTATE_CONSTANT_* with buffer 3 read length equal to zero committed followed by a 3DSTATE_CONSTANT_* with buffer 0 read length not equal to zero committed. Possible ways to avoid this condition include:

- always force buffer 3 to have a non-zero read length
- always force buffer 0 to a zero read length

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 15h 3DSTATE_CONSTANT_VS Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Constant Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command.
	7:0	**DWord Length** Default Value: 9h Excludes DWord (0,1) Format: =n

<table>
<tr><th colspan="3">3DSTATE_CONSTANT_VS</th></tr>
<tr><td>1..10</td><td>319:0</td><td><b>Constant Body</b>
<table><tr><td>Format:</td><td><b>3DSTATE_CONSTANT(Body)</b></td></tr></table>Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS</td></tr>
</table>

3DSTATE_CPS_POINTERS

3DSTATE_CPS_POINTERS		
Source:	RenderCS	
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: Opcode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 22h 3DSTATE_CPS_POINTERS Format: OpCode
	15:0	**DWord Length** Default Value: 0h Format: =n
1	31:0	**CPS Pointers State Body** Format: 3DSTATE_CPS_POINTERS_BODY

3DSTATE_CPSIZE_CONTROL_BUFFER

3DSTATE_CPSIZE_CONTROL_BUFFER

Source:	RenderCS
Length Bias:	2

The CP size Control Buffer surface state is delivered as a pipelined state packet. However, the state change pipelining isn't completely transparent (see restriction below).

WM HW will internally manage the draining pipe and flushing of the caches when this command is issued. The PIPE_CONTROL restrictions are removed.

Programming Notes
If the CPCB surface is not present, SW must set the Surface Type field to SURFTYPE_NULL.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 83h 3DSTATE_CPSIZE_CONTROL_BUFFER Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 6h Excludes Dword (0,1) Format: =n Excludes DWord(0,1)
1..7	223:0	**CPsize Control Buffer Body** Format: 3DSTATE_CPSIZE_CONTROL_BUFFER_BODY

3DSTATE_DEPTH_BOUNDS

3DSTATE_DEPTH_BOUNDS

Source:	BSpec
Length Bias:	2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 03h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: Opcode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: Opcode
	23:16	**3D Command Sub Opcode** Default Value: 71h 3DSTATE_DEPTH_BOUNDS Format: Opcode
	15	**Depth Bounds Test Enable Modify Disable** Format: Disable When this bit is set, the following fields will be ignored: • Depth Bounds Test Enable
	14	**Depth Bounds Test Value Modify Disable** Format: Disable When this bit is set, the following fields will be ignored: • Depth Bounds Test Min Value • Depth Bounds Test Max Value
	13:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 02h Excludes DWord (0,1) Format: =n Total Length - 2
1..3	95:0	**Depth Bounds State Body** Format: **3DSTATE_DEPTH_BOUNDS_BODY**

3DSTATE_DEPTH_BUFFER

3DSTATE_DEPTH_BUFFER

Source:	RenderCS
Length Bias:	2

The depth buffer surface state is delivered as a pipelined state packet. However, the state change pipelining isn't completely transparent (see restriction below).

WM HW will internally manage the draining pipe and flushing of the caches when this command is issued. The PIPE_CONTROL restrictions are removed.

Programming Notes

Note for validation teams. If the depth surface is backdoor initialized or written to directly by the CPU, the values placed in the Depth Surface must be within the numeric range of [0.0 ... 1.0] for DirectX and may in the future include +/- max floating-point values; but not +/-Inf, DNORMs or any NaN code.
If the Depth surface is not present, SW must set the Surface Type field to SURFTYPE_NULL

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 5h 3DSTATE_DEPTH_BUFFER Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Format: =n Excludes DWord(0,1) Value: 6h — Name: Excludes Dword (0,1) **[Default]**

3DSTATE_DEPTH_BUFFER

1

31:29 **Surface Type**

Value	Name	Description
0h	Reserved	Reserved
1h	SURFTYPE_2D	Defines a 2-dimensional map or array of maps
2h	Reserved	Reserved
3h	SURFTYPE_CUBE	Defines a cube map
4h-6h	Reserved	
7h	SURFTYPE_NULL	Defines a null surface

Programming Notes
The Surface Type of the depth buffer must be the same as the Surface Type of the 1. Render target(s) (defined in SURFACE_STATE), unless either the depth buffer or render targets are SURFTYPE_NULL 2. Stencil buffer (defined in 3DSTATE_STENCIL_BUFFER) unless either the depth buffer or Stencil buffer surf_type are SURFTYPE_NULL. If depth is enabled with 1D render target, depth surface type needs to be set to 2D surface type and height set to 1. For this case only, the Surface Type of the depth buffer can be 2D while the Surface Type of the render target(s) are 1D, representing an exception to a programming note above.

28 **Depth Write Enable**

Format:	Enable

This field enables depth writes to the depth buffer surface. Both this field and the Depth Buffer Write Enable field in DEPTH_STENCIL_STATE must be enabled in order for depth writes to occur.

27 **Null Page Coherency Enable**

Format:	Enable

This field is used for enabling NULL coherency as defined under Tiled Resources.

Value	Name
1	Enable
0	Disable **[Default]**

Programming Notes
SW must enable this bit only if Tiled Resource is enabled

26:24 **Surface Format**

Specifies the format of the depth buffer.

Value	Name
0h	Reserved
1h	D32_FLOAT
2h	Reserved
3h	D24_UNORM_X8_UINT

3DSTATE_DEPTH_BUFFER

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>4h</td><td>Reserved</td></tr>
<tr><td>5h</td><td>D16_UNORM</td></tr>
<tr><td>6h-7h</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td></td><td>23</td><td>
<b>Corner Texel Mode</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
This field, when ENABLED, indicates when a surface is using corner texel-mode for depth surface. This bit changes how the size of each MIP when calculating the offset within a surface.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable <b>[Default]</b></td><td>Corner Texel mode is not enabled.</td></tr>
<tr><td>1h</td><td>Enable</td><td>Corner Texel Mode is enabled.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Corner texel for the depth buffer must be the same as the Surface Type of the
1. Render target(s) (defined in SURFACE_STATE), unless either the depth buffer or render targets are SURFTYPE_NULL
2. Stencil buffer (defined in 3DSTATE_STENCIL_BUFFER) unless either the depth buffer or Stencil buffer surf_type are SURFTYPE_NULL</td></tr>
</table>
</td></tr>
<tr><td></td><td>22</td><td>
<b>Hierarchical Depth Buffer Enable</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
If enabled, indicates that a hierarchical depth buffer is defined.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>If this field is enabled, the Software Tiled Rendering Mode must be NORMAL. This field must be disabled if Early Depth Test Enable is disabled OR if depth buffer surface type is NULL. This field must be disabled if surface type field is SURFTYPE_1D</td></tr>
</table>
</td></tr>
<tr><td></td><td>21</td><td>
<b>Depth Buffer Compression Enable</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
if enabled, indicates that Depth Buffer Compression is Enabled
When this field is enabled, Depth Buffer must be initialized via Depth Clear (HZ_OP) when HiZ is enabled. If HiZ is disabled, Depth Buffer must be initialized via full screen primitive with Depth Write enabled and Depth Test Disabled.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>SW must set this bit if the Depth Control surface enable is also set. The depth surface control enable is in Bit[19] of this DWORD.</td></tr>
</table>
</td></tr>
<tr><td></td><td>20</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th colspan="3">3DSTATE_DEPTH_BUFFER</th></tr>
<tr><td></td><td>19</td><td><b>Control Surface Enable</b>
If set to 1, it indicates if the common control surface is present. The read and write transaction opcodes sent by RCZ to the fabric are different depending on the control surface. If the control surface is not present, the reads and writes are in legacy mode. If the control surface is present, the reads and write opcodes will be either UNCOMPRESSED_TYP for uncompressible transactions (resolves) or COMPRESSED_TYP for compressible transactions.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>SW must set this bit to "1", if the common control surface is present in the system.</td></tr>
</table></td></tr>
<tr><td></td><td>18</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>17:0</td><td><b>Surface Pitch</b>
<table>
<tr><td>Format:</td><td>U18-1</td></tr>
</table>
For TileYF and TileYS surfaces, the range is dependent on the Cu parameter (refer to <i>Memory Data Formats</i> section for the definition of the Cu parameter depending on the case). The range in bytes is [2^{Cu}-1,262143] -> [(2^{Cu})B,256KB] = [1 tile, 256KB/(2^{Cu}) tiles]
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>[7Fh,3FFFFh]</td><td></td><td>corresponding to [128B, 256KB] also restricted to a multiple of 128B</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td><i>The minimum pitch should be calculated based on Cu, Cv, W_L.
The Cu, Cv are the tile constants and $W_L$is the aligned width adjusted for MSAA.
Refer to 2D Surfaces to get the Cu, Cv, $W_L$values and Calculations.
Then use this for pitch formula:
Minimum_pitch =(ceiling((W_0* pixel_size) / (1 « Cu)) *(1 « Cu)) 1 ; //$W_0$is the aligned width for the largest LOD (i.e LOD 0)
(1 « Cu) = tile width in bytes
(1 « Cv) = tile height in lines</i></td></tr>
</table></td></tr>
<tr><td>2..3</td><td>63:0</td><td><b>Surface Base Address</b>
<table>
<tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This field specifies address of the buffer in mapped Graphics Memory. The Depth Buffer can only be mapped to Main Memory (uncached). If the surface is tiled, the base address must conform to the Per-Surface Tiling Alignment. If the buffer is linear, the surface must be 64-byte aligned.
If the buffer is linear, the surface must be 64-byte aligned.</td></tr>
</table></td></tr>
<tr><td>4</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

3DSTATE_DEPTH_BUFFER

30:17 **Height**

Format:	U14-1

This field specifies the height of the surface. If the surface is MIP-mapped, this field contains the height of the base MIP level.

Value	Name	Description	Exists If
[0,16383]	Legal Range	Height of surface - 1 (y/v dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D')
[0,16383]	Legal Range	y/v dimension	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_CUBE')

Programming Notes
The Height of the depth buffer must be the same as the Height of the 1. render target(s) (defined in SURFACE_STATE), unless Surface Type is SURFTYPE_2D with Depth = 0 (non-array) and LOD = 0 (non-mip mapped) or if SURFACE_STATE_SURFTYPE is NULL. 2.Stencil buffer (defined in 3DSTATE_STENCIL_BUFFER) unless Stencil buffer surf_type is SURFTYPE_NULL

16 **Reserved**

Access:	RO
Format:	MBZ

15 **Reserved**

Access:	RO
Format:	MBZ

14:1 **Width**

Format:	U14-1

This field specifies the width of the surface. If the surface is MIP-mapped, this field specifies the width of the base MIP level. The width is specified in units of pixels.

Value	Name	Description	Exists If
[0,16383]	Legal Range	Width of surface - 1 (x/u dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D')
[0,16383]	Legal Range	Width of surface - 1 (x/u dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_CUBE')

3DSTATE_DEPTH_BUFFER

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>The Width specified by this field must be less than or equal to the surface pitch (specified in bytes via the Surface Pitch field). For cube maps, Width must be set equal to Height.
The Width of the depth buffer must be the same as the
1. Width of the render target(s) (defined in SURFACE_STATE), unless Surface Type is SURFTYPE_1D or SURFTYPE_2D with Depth = 0 (non-array) and LOD = 0 (non-mip mapped) or if SURFACE_STATE_SURFTYPE is NULL.
2.Stencil buffer (defined in 3DSTATE_STENCIL_BUFFER) unless Stencil buffer surf_type is SURFTYPE_NULL</td></tr>
</table>
</td></tr>
<tr><td></td><td>0</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>5</td><td>31</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>30:20</td><td>
<b>Depth</b>
<table>
<tr><td>Format:</td><td>U11-1</td></tr>
</table>
This field specifies the total number of levels for a volume texture or the number of array elements allowed to be accessed starting at the Minimum Array Element for arrayed surfaces. If the volume texture is MIP-mapped, this field specifies the depth of the base MIP level.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Exists If</th></tr>
<tr><td>[0,2047]</td><td>Legal Range</td><td>Number of array elements - 1</td><td>(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D')</td></tr>
<tr><td>[0,0]</td><td>Legal Range</td><td>Must be zero</td><td>(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_CUBE')</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>The Depth of the depth buffer must be the same as
1. The Depth of the render target(s) (defined in SURFACE_STATE).
2.Stencil buffer (defined in 3DSTATE_STENCIL_BUFFER) unless Stencil buffer surf_type is SURFTYPE_NULL</td></tr>
</table>
</td></tr>
<tr><td></td><td>19</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>18:8</td><td>
<b>Minimum Array Element</b>
<table>
<tr><td>Format:</td><td>U11</td></tr>
</table>
For 2D Surfaces: This field indicates the minimum array element that can be accessed as part of this surface. The delivered array index is added to this field before being used to address the surface. For Other Surfaces This field is ignored
</td></tr>
</table>

3DSTATE_DEPTH_BUFFER

Value	Name	Exists If
[0,2047]	SURFTYPE_2D	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_1D'|Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D')

Programming Notes
Minimum array element of the depth buffer must be the same as the Surface Type of the 1. Render target(s) (defined in SURFACE_STATE), unless either the depth buffer or render targets are SURFTYPE_NULL 2. Stencil buffer (defined in 3DSTATE_STENCIL_BUFFER) unless either the depth buffer or Stencil buffer surf_type are SURFTYPE_NULL

7

Reserved

Access:	RO
Format:	MBZ

6:0

Depth Buffer Object Control State

Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for the depth buffer.

6

31:30

Tiled Mode

For Depth Buffer Surfaces: This field specifies the tiled mode. For other surfaces: This field is ignored.

Value	Name
0h	Reserved
1h	Tile64
2h	Reserved
3h	Tile4

Programming Notes
If **Tile Mode** is not set to TILEMODE_YMAJOR, this field must be set to TRMODE_NONE.

29:26

Mip Tail Start LOD

Format:	U4

For Sampling Engine, Render Target, and Typed Surfaces: This field indicates which LOD is the first one in the MIP tail if **Tiled Mode** is not TRMODE_NONE. The MIP tail has a different layout than the rest of the surface. Refer to the *Memory Data Formats* section for more details.
For other surfaces: This field is ignored.

Programming Notes
This field must be zero if the **Surface Format** is MONO8. This field is ignored if **Tiled Mode** is TRMODE_NONE unless **Surface Type** is SURFTYPE_1D.
If Tiled Mode is not TRMODE_NONE, this field must be set to ensure that mips within the mip tail do not overlap given the storage algorithms given in the Memory Data Formats section. The following table indicates the maximum size of the mip that is set to be the Mip Tail Start LOD

3DSTATE_DEPTH_BUFFER

for various cases:

Tiling Mode	Slot Size in Bytes	8-bit Size	16-bit Size	32-bit Size
2D TIleYs 1x	32KB	(128, 256)	(128, 128)	(64, 128)
2D TileYf 1x	2KB	(32, 64)	(32, 32)	(16, 32)

25:6 **Reserved**

Access:	RO
Format:	MBZ

5 **Compression Mode**

Specifies whether HW should choose hardcoded encodings (disabled) or SW programmable encoding defined in [4:0] (enabled).

Value	Name	Description
0h	Disable **[Default]**	Use hardcoded (legacy) encodings based on surface format.
1h	Enable	Use SW programmable encodings defined in DWord6 [4:0]

4:0 **Render Compression Format**

Format:	**Render Compression Format**

Specifies the 5 bit compression format.

7 31:21 **Render Target View Extent**

Format:	U11-1

Value	Name	Description	Exists If
[0,2047]	Legal Range	Number of array elements- 1	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D')
[0,0]	Legal Range	Must be zero	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_CUBE')

Programming Notes
Render target View Extent of the depth buffer must be the same as the Surface Type of the 1. Render target(s) (defined in SURFACE_STATE), unless either the depth buffer or render targets are SURFTYPE_NULL 2. Stencil buffer (defined in 3DSTATE_STENCIL_BUFFER) unless either the depth buffer or Stencil buffer surf_type are SURFTYPE_NULL

20 **Reserved**

Format:	MBZ

19:16 **Surf LOD**

Value	Name
[0-14]	

3DSTATE_DEPTH_BUFFER

Programming Notes
Surf LOD of the depth buffer must be the same as the Surface Type of the 1. Render target(s) (defined in SURFACE_STATE), unless either the depth buffer or render targets are SURFTYPE_NULL 2. Stencil buffer (defined in 3DSTATE_STENCIL_BUFFER) unless either the depth buffer or Stencil buffer surf_type are SURFTYPE_NULL

15 **Reserved**

Access:	RO
Format:	MBZ

14:0 **Surface QPitch**

Format:	U17[16:2]

Format: QPitch[16:2] The interpretation of this field is dependent on **Surface Type** as follows: • SURFTYPE_2D/CUBE: distance in *rows* between array slices.
Other surface types: field is ignored

Value	Name	Description
[1h,7FFFh]		in multiples of 4 (low 2 bits missing)

Programming Notes
For 2D Surfaces: This field must be set to the same value as the Depth field. For Other Surfaces, This field is ignored . Refer to Alignment Unit Size for alignment sizes based on MSAA and Depth-format. Software must ensure that this field is set to a value sufficiently large that array slices in the surface do not overlap. Refer to the Memory Data Formats section for information on how surfaces are stored. TYS/TYF QPitch is valid only for 2D array surfaces and represents the tile-padded total number of texels(lines) in a single array slice. Height of each LOD: HL = AlignToTileHeight(MSAA_height_factor * (**height**»L) > 0?**height**»L : 1), where AlignToTileHeight(x) is (ceiling((x) / (1 « Cv)) *(1 « Cv)) Height of all LODs is a sum: H = H0+H1+..Hn, N is number of mip levels. If surface has MIP tail, equation stops at Hn where n=MipTailStartLOD. MipTail is single tile. QPitch is multiple of tile height (1 « Cv) and should be equal or greater H computed above.

8..9 63:0 **Reserved**

Format:	MBZ

3DSTATE_DRAWING_RECTANGLE

<table>
<tr><th colspan="3">3DSTATE_DRAWING_RECTANGLE</th></tr>
<tr><td colspan="3">Source: RenderCS
Length Bias: 2</td></tr>
<tr><td colspan="3">The 3DSTATE_DRAWING_RECTANGLE command is used to set the 3D drawing rectangle and related state.</td></tr>
<tr><td colspan="3"><table><tr><th>Restriction</th></tr><tr><td>When executed in POCS command stream, this programs the drawing rectangle for the SFR stage of the POCS pipeline.</td></tr></table></td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h GFXPIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Command SubType</b><table><tr><td>Default Value:</td><td>3h GFXPIPE_3D</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><b>3D Command Opcode</b><table><tr><td>Default Value:</td><td>1h 3DSTATE_NONPIPELINED</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>3D Command Sub Opcode</b><table><tr><td>Default Value:</td><td>00h 3DSTATE_DRAWING_RECTANGLE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:14</td><td><b>Core Mode Select</b><table><tr><td>Format:</td><td>U2</td></tr></table>Specifies which core this command will be considered valid and update based on the state in this command.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Legacy</td><td>Both cores are enabled and will update the state.</td></tr><tr><td>1h</td><td>Core 0 Enabled</td><td>State will be updated in Core 0 only</td></tr><tr><td>2h</td><td>Core 1 Enabled</td><td>State will be updated in Core 1 only</td></tr><tr><td>3h</td><td>Reserved</td><td></td></tr></table></td></tr>
<tr><td></td><td>13:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b><table><tr><td>Default Value:</td><td>2h Excludes DWord (0,1)</td></tr><tr><td>Format:</td><td>=n</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">3DSTATE_DRAWING_RECTANGLE</th></tr>
<tr><td>1</td><td>31:16</td><td><b>Clipped Drawing Rectangle Y Min</b>
<table><tr><td>Format:</td><td>U16</td></tr></table>
Specifies Ymin value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with Y coordinates less than Ymin will be clipped out.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,16383]</td><td>Device ignores bits 31:30</td></tr></table>
<table><tr><th>Programming Notes</th></tr><tr><td>This value can be larger than Clipped Drawing Rectangle Y Max. If Ymin>Ymax, the clipped drawing rectangle is null, all polygons are discarded. If Ymin==Ymax, the clipped drawing rectangle is 1 pixel wide in the Y direction.</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Clipped Drawing Rectangle X Min</b>
<table><tr><td>Format:</td><td>U16</td></tr></table>
Specifies Xmin value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with X coordinates less than Xmin will be clipped out.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,16383]</td><td>Device ignores bits 15:14</td></tr></table>
<table><tr><th>Programming Notes</th></tr><tr><td>This value can be larger than Clipped Drawing Rectangle X Max. If Xmin>Xmax, the clipped drawing rectangle is null, all polygons are discarded. If Xmin==Xmax, the clipped drawing rectangle is 1 pixel wide in the X direction.</td></tr></table></td></tr>
<tr><td>2</td><td>31:16</td><td><b>Clipped Drawing Rectangle Y Max</b>
<table><tr><td>Format:</td><td>U16</td></tr></table>
Specifies Ymax value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with coordinates greater than Ymax will be clipped out.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,16383]</td><td>Device ignores bits 31:30</td></tr></table>
<table><tr><th>Programming Notes</th></tr><tr><td>This value can be less than Clipped Drawing Rectangle Y Min. If Ymax<Ymin, the clipped drawing rectangle is null, all polygons are discarded. If Ymin==Ymax, the clipped drawing rectangle is 1 pixel wide in the Y direction.</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Clipped Drawing Rectangle X Max</b>
<table><tr><td>Format:</td><td>U16</td></tr></table>
Specifies Xmax value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with coordinates greater than Xmax will be clipped out.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,16383]</td><td>Device ignores bits 15:14</td></tr></table></td></tr>
</table>

3DSTATE_DRAWING_RECTANGLE

		Programming Notes This value can be less than Clipped Drawing Rectangle X Min. If Xmax<Xmin, the clipped drawing rectangle is null, all polygons are discarded. If Xmin==Xmax, the clipped drawing rectangle is 1 pixel wide in the X direction.
3	31:16	**Drawing Rectangle Origin Y** Format: S15 Range: [-16384,16383] (Bit 31 should be a sign extension) Specifies Y origin of Drawing Rectangle (in whole pixels) relative to origin of the Color Buffer, used to map incoming (Draw Rectangle-relative) vertex positions to the Color Buffer space.
	15:0	**Drawing Rectangle Origin X** Format: S15 Range: [-16384,16383] (Bit 15 should be a sign extension) Specifies X origin of Drawing Rectangle (in whole pixels) relative to origin of the Color Buffer, used to map incoming (Draw Rectangle-relative) vertex positions to the Color Buffer space.

3DSTATE_DS

3DSTATE_DS

Source:	RenderCS
Length Bias:	2

The state used by DS is defined with this inline state packet

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Dh 3DSTATE_DS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 9h Excludes DWord (0,1) Format: =n
1..10	319:0	**DS State Body** Format: **3DSTATE_DS_BODY**

3DSTATE_GS

3DSTATE_GS		
Source:	RenderCS	
Length Bias:	2	
Controls the GS stage hardware.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 11h 3DSTATE_GS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 8h Excludes DWord (0,1) Format: =n
1..9	287:0	**GS State Body** Format: 3DSTATE_GS_BODY

3DSTATE_HIER_DEPTH_BUFFER

3DSTATE_HIER_DEPTH_BUFFER

Source:	RenderCS
Length Bias:	2

This command sets the surface state of the hierarchical depth buffer, delivered as a pipelined state command. However, the state change pipelining isn't completely transparent (see restriction below).

WM HW will internally manage the draining pipe and flushing of the caches when this command is issued. The PIPE_CONTROL restrictions are removed.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 07h 3DSTATE_HIER_DEPTH_BUFFER Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**Dword Length** Default Value: 3h Excludes Dword (0,1) Format: =n
1..4	127:0	**Hier Depth Buffer State Body** Format: 3DSTATE_HIER_DEPTH_BUFFER_BODY

3DSTATE_HS

3DSTATE_HS

Source:	RenderCS
Length Bias:	2

Controls the HS stage hardware.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Bh 3DSTATE_HS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 7 Excludes DWord (0,1) Format: =n
1..8	255:0	**HS State Body** Format: **3DSTATE_HS_BODY**

3DSTATE_INDEX_BUFFER

3DSTATE_INDEX_BUFFER

Source:	RenderCS
Length Bias:	2

This command is used to specify the current IB state used by the VF function. At most one IB is defined and active at any given time. NOTES: The IB must be specified before any RANDOM 3D_PRIMITIVE commands are issued It is possible to have vertex elements source completely from generated ID values and therefore not require any Index Buffer accesses. In this case, VF function will simply ignore the Index Buffer state.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Ah 3DSTATE_INDEX_BUFFER Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 3h Excludes DWord (0,1) Format: =n
1..4	127:0	**Index Buffer State Body** Format: 3DSTATE_INDEX_BUFFER_BODY

3DSTATE_LINE_STIPPLE

3DSTATE_LINE_STIPPLE

Source:	RenderCS
Length Bias:	2

The 3DSTATE_LINE_STIPPLE command is used to specify state variables used in the Line Stipple function.

Workaround
Workaround : This command must be followed by a PIPE_CONTROL with CS Stall bit set.,

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 08h 3DSTATE_LINE_STIPPLE Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**Dword Length** Default Value: 1h Excludes Dword (0,1) Format: =n
1	31	**Modify Enable (Current Repeat Counter, Current Stipple Index)** Format: Enable Modify enable for **Current Repeat Counter** and **Current Stipple Index** fields. **Programming Notes** It is provided only for HW-generated commands as part of context save/restore. SW must initialize the current repeat counter, current stipple count fields if it sets this bit to enable. SW must set this bit to reset the stipple count.
	30	**Reserved** Access: RO Format: MBZ

3DSTATE_LINE_STIPPLE		
	29:21	**Current Repeat Counter** Format: U9 This field sets the HW-internal repeat counter state. SW must initialize it to 1 if the modify enable is set.
	20	**Reserved** Access: RO Format: MBZ
	19:16	**Current Stipple Index** Format: U4 This field sets the HW-internal stipple pattern index. SW must initialize it to 0 if the modify enable is set.
	15:0	**Line Stipple Pattern** Format: Enable[16] Specifies a pattern used to mask out bit specific pixels while rendering lines.
2	31:15	**Line Stipple Inverse Repeat Count** Format: U1.16 Range: [0.00390625, 1.0] Specifies the inverse (truncated) of the repeat count for the line stipple function.
	14:9	**Reserved** Access: RO Format: MBZ
	8:0	**Line Stipple Repeat Count** Format: U9 Specifies the repeat count for the line stipple function. **Value** / **Name** [1, 256] /

3DSTATE_MESH_CONTROL

3DSTATE_MESH_CONTROL - 3DSTATE_MESH_CONTROL		
Source: BSpec Length Bias: 2		
Specifies low-frequency control states for the MeshShader stage.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 77h 3DSTATE_MESH_CONTROL Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n
1..2	63:0	**Mesh Shader Control Body** Format: 3DSTATE_MESH_CONTROL_BODY

3DSTATE_MESH_DISTRIB

3DSTATE_MESH_DISTRIB - 3DSTATE_MESH_DISTRIB

Source:	BSpec
Length Bias:	2

Specifies states that control distribution of 3DMESH commands across geometry pipelines.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 78h 3DSTATE_MESH_DISTRIB Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n
1	31:0	**Mesh Shader Distrib Body** Format: **3DSTATE_MESH_DISTRIB_BODY**

3DSTATE_MESH_SHADER_DATA

3DSTATE_MESH_SHADER_DATA - 3DSTATE_MESH_SHADER_DATA

Source:	BSpec
Length Bias:	2

Specifies data-input states for the MeshShader stage.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 7Bh 3DSTATE_MESH_SHADER_DATA Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 8h Excludes DWord (0,1) Format: =n
1..9	287:0	**Mesh Shader Data Body** Format: 3DSTATE_MESH_SHADER_DATA_BODY

3DSTATE_MESH_SHADER

3DSTATE_MESH_SHADER - 3DSTATE_MESH_SHADER

Source:	BSpec
Length Bias:	2

Specifies shader-related states for the MeshShader stage.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 7Ah 3DSTATE_MESH_SHADER Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 6h Excludes DWord (0,1) Format: =n
1..7	223:0	**Mesh Shader Body** Format: 3DSTATE_MESH_SHADER_BODY

3DSTATE_MULTISAMPLE

3DSTATE_MULTISAMPLE

Source:	RenderCS
Length Bias:	2

The 3DSTATE_MULTISAMPLE command is used to specify multisample state associated with the current render target/depth buffer/stencil buffer.

Programming Notes
It is illegal to render to surfaces with multiple different values of the state fields in this command.
Restriction : When executed in the POCS command stream, this command programs the multisample state for the Raster stage of the POCS pipeline

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Dh 3DSTATE_MULTISAMPLE Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n
1	31:0	**Multisample State Body** Format: 3DSTATE_MULTISAMPLE_BODY

3DSTATE_POLY_STIPPLE_OFFSET

3DSTATE_POLY_STIPPLE_OFFSET

Source:	RenderCS
Length Bias:	2

The 3DSTATE_POLY_STIPPLE_OFFSET command is used to specify the origin of the repeated screen-space Polygon Stipple Pattern as an X, Y offset from the Color Buffer origin.

Workaround
Workaround : This command must be followed by a PIPE_CONTROL with CS Stall bit set.,

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h GFXPIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Command SubType</b><table><tr><td>Default Value:</td><td>3h GFXPIPE_3D</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><b>3D Command Opcode</b><table><tr><td>Default Value:</td><td>1h 3DSTATE_NONPIPELINED</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>3D Command Sub Opcode</b><table><tr><td>Default Value:</td><td>06h 3DSTATE_POLY_STIPPLE_OFFSET</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>Dword Length</b><table><tr><td>Default Value:</td><td>0h Excludes Dword (0,1)</td></tr><tr><td>Format:</td><td>=n</td></tr></table></td></tr>
<tr><td>1</td><td>31:13</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>12:8</td><td><b>Polygon Stipple X Offset</b><table><tr><td>Format:</td><td>U5</td></tr></table>Specifies a 5 bit x address offset in the poly stipple pattern<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,31]</td><td></td></tr></table></td></tr>
</table>

3DSTATE_POLY_STIPPLE_OFFSET

<table>
<tr><th colspan="3">3DSTATE_POLY_STIPPLE_OFFSET</th></tr>
<tr><td></td><td>7:5</td><td><b>Reserved</b>
Access: RO
Format: MBZ</td></tr>
<tr><td></td><td>4:0</td><td><b>Polygon Stipple Y Offset</b>
Format: U5
Specifies a 5 bit y address offset in the poly stipple pattern
Value | Name
[0,31] |</td></tr>
</table>

3DSTATE_POLY_STIPPLE_PATTERN

3DSTATE_POLY_STIPPLE_PATTERN		
Source:	RenderCS	
Length Bias:	2	
The 3DSTATE_POLY_STIPPLE_PATTERN command is used to specify the 32x32 Polygon Stipple Pattern used in the Polygon Stipple function of the WM unit.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 07h 3DSTATE_POLY_STIPPLE_PATTERN Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**Dword Length** Default Value: 1Fh Excludes Dword (0,1) Format: =n
1..32	1023:0	**Pattern Row** Format: U32[32] Specifies a pattern used by Polygon Stipple to mask out specific pixels of every 32x32 area rendered.

3DSTATE_PRIMITIVE_REPLICATION

3DSTATE_PRIMITIVE_REPLICATION

Source:	RenderCS, PositionCS
Length Bias:	2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 6Ch 3DSTATE_PRIMITIVE_REPLICATION Format: OpCode
	15:11	**Reserved** Access: RO Format: MBZ
	10	**Prim Rep ViewPort Offsets Disable** Format: Disable When this bit is set, the following fields will be ignored: • ViewPort Offsets • RTAI Offsets
	9	**Prim Rep Replication Count Disable** Format: Disable When this bit is set, the following fields will be ignored: • Replication Count
	8	**Prim Rep Replica Mask Disable** Format: Disable When this bit is set, the following fields will be ignored: • Replica Mask
	7:0	**DWord Length** Default Value: 4h Excludes DWord (0,1) Format: =n

3DSTATE_PRIMITIVE_REPLICATION		
1..5	159:0	**Primitive Replication State Body** Format: 3DSTATE_PRIMITIVE_REPLICATION_BODY

3DSTATE_PS_BLEND

3DSTATE_PS_BLEND

Source:	RenderCS
Length Bias:	2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4Dh 3DSTATE_PS_BLEND Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:0	**PS Blend State Body** Format: 3DSTATE_PS_BLEND_BODY

3DSTATE_PS_EXTRA

3DSTATE_PS_EXTRA

Source:	RenderCS
Length Bias:	2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4fh 3DSTATE_PS_EXTRA Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:0	**PS Extra State Body** Format: 3DSTATE_PS_EXTRA_BODY

3DSTATE_PS

3DSTATE_PS

Source:	RenderCS
Length Bias:	2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 20h 3DSTATE_PS Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0Ah Excludes DWord (0,1) Format: =n
1..11	351:0	**PS State Body** Format: 3DSTATE_PS_BODY

3DSTATE_PUSH_CONSTANT_ALLOC_DS

3DSTATE_PUSH_CONSTANT_ALLOC_DS

Source:	RenderCS
Length Bias:	2

This command sets up the URB configuration for DS Push Constant Buffer.

Programming Notes

Programming Restriction:

- The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.
- The sum of the constant length of the push constants must be equal or smaller then the size of the allocated space in the URB including the buffering for half cachelines. See **Push Constant URB Allocation section for more details**.
- The Domain Shader Push Constants state must be programmed prior to any state is committed into the pipeline or any preemptible command(i.e. prior to PIPE_CONTROL, 3DPRIMITIVE, 3DMESH or COMPUTE_WALKER)after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_DS.

When gather at set shader is disabled, Push Constant command is committed when 3DPRIMITIVE command is parsed.
When gather at set shader is enabled, commit point on the Push Constant command is a 3DSTATE_BINDING_TABLE_POINTER command.

The 3DSTATE_BINDING_TABLE_POINTER must be reprogrammed after 3DSTATE_PUSH_CONST_ALLOC command and the reprogramming of the push constants prior to the next 3DPRIMITIVE if gather at set shader is enabled to ensure the new programming is committed.

DWord	Bit	Description
0 **Programming Notes:** This command must be followed by a PIPE_CONTROL with CS Stall bit set.,	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 14h 3DSTATE_PUSH_CONSTANT_ALLOC_DS Format: OpCode

3DSTATE_PUSH_CONSTANT_ALLOC_DS

	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n
1	31:21	**Reserved** Access: RO Format: MBZ
	20:16	**Constant Buffer Offset** Format: U5 Specifies the offset of the DS constant buffer into the URB. Value: [0,31] — Name: (0KB - 31KB) Increments of 2KB
	15:6	**Reserved** Access: RO Format: MBZ
	5:0	**Constant Buffer Size** Format: U6 Specifies the size of the DS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for DS. Value: [0,32] — Name: (0KB - 32KB) Increments of 2KB

3DSTATE_PUSH_CONSTANT_ALLOC_GS

3DSTATE_PUSH_CONSTANT_ALLOC_GS

Source:	RenderCS
Length Bias:	2

This command sets up the URB configuration for GS Push Constant Buffer.

Programming Notes

- The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.
- The sum of the constant length of the push constants must be equal or smaller then the size of the allocated space in the URB including the buffering for half cachelines. See **Push Constant URB Allocation section for more details**.
- The Geometry Shader Push Constants state must be programmed prior to any state is committed into the pipeline or any preemptible command(i.e. prior to PIPE_CONTROL, 3DPRIMITIVE, 3DMESH or COMPUTE_WALKER)after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_GS.

When gather at set shader is disabled, Push Constant command is committed when 3DPRIMITIVE command is parsed.
When gather at set shader is enabled, commit point on the Push Constant command is a 3DSTATE_BINDING_TABLE_POINTER command.

The 3DSTATE_BINDING_TABLE_POINTER must be reprogrammed after 3DSTATE_PUSH_CONST_ALLOC command and the reprogramming of the push constants prior to the next 3DPRIMITIVE if gather at set shader is enabled to ensure the new programming is committed.

Workaround

This command must be followed by a PIPE_CONTROL with CS Stall bit set.,

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 15h 3DSTATE_PUSH_CONSTANT_ALLOC_GS Format: OpCode

3DSTATE_PUSH_CONSTANT_ALLOC_GS

<table>
<tr><td></td><td>15:8</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Format:</td><td>=n</td></tr>
</table>
Total Length - 2
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>3DSTATE_PUSH_CONSTANT_ALLOC_GS <b>[Default]</b></td><td>Excludes DWord (0,1)</td></tr>
</table></td></tr>
<tr><td>1</td><td>31:21</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>20:16</td><td><b>Constant Buffer Offset</b>
<table>
<tr><td>Format:</td><td>U5</td></tr>
</table>
Specifies the offset of the GS constant buffer into the URB.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>[0,31]</td><td>(0KB - 31KB) Increments of 2KB</td></tr>
</table></td></tr>
<tr><td></td><td>15:6</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>5:0</td><td><b>Constant Buffer Size</b>
<table>
<tr><td>Format:</td><td>U6</td></tr>
</table>
Specifies the size of the GS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for GS.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>[0,32]</td><td>(0KB - 32KB) Increments of 2KB</td></tr>
</table></td></tr>
</table>

3DSTATE_PUSH_CONSTANT_ALLOC_HS

3DSTATE_PUSH_CONSTANT_ALLOC_HS

Source:	RenderCS
Length Bias:	2

This command sets up the URB configuration for HS Push Constant Buffer.

Programming Notes

Programming Restriction:

- The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.
- The sum of the constant length of the push constants must be equal or smaller than the size of the allocated space in the URB including the buffering for half cachelines. See **Push Constant URB Allocation section for more details**.
- The Hull Shader Push Constants state must be programmed prior to any state is committed into the pipeline or any preemptible command(i.e. prior to PIPE_CONTROL, 3DPRIMITIVE, 3DMESH or COMPUTE_WALKER)after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_HS.

When gather at set shader is disabled, Push Constant command is committed when 3DPRIMITIVE command is parsed.
When gather at set shader is enabled, commit point on the Push Constant command is a 3DSTATE_BINDING_TABLE_POINTER command.

The 3DSTATE_BINDING_TABLE_POINTER must be reprogrammed after 3DSTATE_PUSH_CONST_ALLOC command and the reprogramming of the push constants prior to the next 3DPRIMITIVE if gather at set shader is enabled to ensure the new programming is committed.

Workaround

This command must be followed by a PIPE_CONTROL with CS Stall bit set.,

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 13h 3DSTATE_PUSH_CONSTANT_ALLOC_HS Format: OpCode

<table>
<tr><th colspan="3">3DSTATE_PUSH_CONSTANT_ALLOC_HS</th></tr>
<tr><td></td><td>15:8</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table><tr><td>Default Value:</td><td>0h Excludes DWord (0,1)</td></tr><tr><td>Format:</td><td>=n</td></tr></table></td></tr>
<tr><td>1</td><td>31:21</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>20:16</td><td><b>Constant Buffer Offset</b>
<table><tr><td>Format:</td><td>U5</td></tr></table>
Specifies the offset of the HS constant buffer into the URB.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,31]</td><td>(0KB - 31KB) Increments of 2KB</td></tr></table></td></tr>
<tr><td></td><td>15:6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Constant Buffer Size</b>
<table><tr><td>Format:</td><td>U6</td></tr></table>
Specifies the size of the HS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for HS.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,32]</td><td>(0KB - 32KB) Increments of 2KB</td></tr></table></td></tr>
</table>

3DSTATE_PUSH_CONSTANT_ALLOC_PS

3DSTATE_PUSH_CONSTANT_ALLOC_PS

Source:	RenderCS
Length Bias:	2

This command sets up the URB configuration for PS Push Constant Buffer.

Programming Notes

Restriction:

- The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.
- The sum of the constant length of the push constants must be equal or smaller then the size of the allocated space in the URB including the buffering for half cachelines. See **Push Constant URB Allocation section for more details**.
- The Pixel Shader Push Constants state must be programmed prior to any state is committed into the pipeline or any preemptible command(i.e. prior to PIPE_CONTROL, 3DPRIMITIVE, 3DMESH or COMPUTE_WALKER)after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_PS.

When gather at set shader is disabled, Push Constant command is committed when 3DPRIMITIVE command is parsed.
When gather at set shader is enabled, commit point on the Push Constant command is a 3DSTATE_BINDING_TABLE_POINTER command.

The 3DSTATE_BINDING_TABLE_POINTER must be reprogrammed after 3DSTATE_PUSH_CONST_ALLOC command and the reprogramming of the push constants prior to the next 3DPRIMITIVE if gather at set shader is enabled to ensure the new programming is committed.

Workaround

This command must be followed by a PIPE_CONTROL with CS Stall bit set.,

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 16h 3DSTATE_PUSH_CONSTANT_ALLOC_PS Format: OpCode

<table>
<tr><th colspan="3">3DSTATE_PUSH_CONSTANT_ALLOC_PS</th></tr>
<tr><td></td><td>15:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>Dword Length</b><table><tr><td>Default Value:</td><td>0h Excludes Dword (0,1)</td></tr><tr><td>Format:</td><td>=n</td></tr></table></td></tr>
<tr><td>1</td><td>31:21</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>20:16</td><td><b>Constant Buffer Offset</b><table><tr><td>Format:</td><td>U5</td></tr></table>Specifies the offset of the PS constant buffer into the URB.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,31]</td><td>(0KB - 31KB) Increments of 2KB</td></tr></table></td></tr>
<tr><td></td><td>15:6</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Constant Buffer Size</b><table><tr><td>Format:</td><td>U6</td></tr></table>Specifies the size of the PS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for PS.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,32]</td><td>(0KB - 32KB) Increments of 2KB</td></tr></table></td></tr>
</table>

3DSTATE_PUSH_CONSTANT_ALLOC_VS

3DSTATE_PUSH_CONSTANT_ALLOC_VS

Source:	RenderCS
Length Bias:	2

This command sets up the URB configuration for VS Push Constant Buffer.

Programming Notes

Programming Restriction:

- The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.
- The sum of the constant length of the push constants must be equal or smaller then the size of the allocated space in the URB including the buffering for half cachelines. See **Push Constant URB Allocation section for more details**.
- The Vertex Shader Push Constants state must be programmed prior to any state is committed into the pipeline or any preemptible command(i.e. prior to PIPE_CONTROL, 3DPRIMITIVE, 3DMESH or COMPUTE_WALKER)after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_VS.

When gather at set shader is disabled, programmed constants are committed when 3DPRIMITIVE command is parsed.
When gather at set shader is enabled, commit point of the constants programmed area 3DSTATE_BINDING_TABLE_POINTER command.

The 3DSTATE_BINDING_TABLE_POINTER must be reprogrammed after 3DSTATE_PUSH_CONST_ALLOC command and the reprogramming of the push constants prior to the next 3DPRIMITIVE if gather at set shader is enabled to ensure the new programming is committed.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 12h 3DSTATE_PUSH_CONSTANT_ALLOC_VS Format: OpCode

3DSTATE_PUSH_CONSTANT_ALLOC_VS

	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n
1	31:21	**Reserved** Access: RO Format: MBZ
	20:16	**Constant Buffer Offset** Format: U5 Specifies the offset of the VS constant buffer into the URB. **Value** | **Name** [0,31] | (0KB - 31KB) Increments of 2KB **Programming Notes** When executed from the POCS pipe, the offset is relative to the VSR_PUSH_CONSTANT_BASE (MMIO offset e518)region reserved for POCS pipe Push Constants.
	15:6	**Reserved** Access: RO Format: MBZ
	5:0	**Constant Buffer Size** Format: U6 Specifies the size of the VS constant buffer. This value will determine the amount of data the command stream can pre-fetch before the buffer is full. Value of zero is only valid when constants are not enabled for VS. **Value** | **Name** [0,32] | (0KB - 32KB) Increments of 2KB

3DSTATE_RASTER

3DSTATE_RASTER

Source:	RenderCS
Length Bias:	2

Restriction
When executed in the POCS command stream, this command programs the raster state for CLR and SFR stages of the POCS pipeline

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 50h 3DSTATE_RASTER Format: OpCode
	15:14	**Reserved** Access: RO Format: MBZ
	13	**Raster State Modify Disable** Format: Disable When this bit is set, the following fields will be ignored: • SubPixel Aligned Quad Rasterization Enable • Line/Point Conservative Rasterization Enable • Conservative Rasterization Enable • API Mode • Forced Sample Count • Force Multisampling • Smooth Point Enable • DX Multisample Rasterization Enable • DX Multisample Rasterization Mode • Global Depth Offset Enable Solid • Global Depth Offset Enable Wireframe

3DSTATE_RASTER

		• Global Depth Offset Enable Point • Antialiasing Enable • Scissor Rectangle Enable
	12	**Front Winding Modify Disable** Format: Disable When this bit is set, the following fields will be ignored: • Front Winding
	11	**Cull Mode Modify Disable** Format: Disable When this bit is set, the following fields will be ignored: • Cull Mode
	10	**Fill Mode Modify Disable** Format: Disable When this bit is set, the following fields will be ignored: • Front Face Fill Mode • Back Face Fill Mode
	9	**Viewport Z Clip Test Enable Modify Disable** Format: Disable When this bit is set, the following fields will be ignored: • Viewport Z Far Clip Test Enable • Viewport Z Near Clip Test Enable
	8	**Global Depth Offset Modify Disable** Format: Disable When this bit is set, the following fields will be ignored: • Global Depth Offset Constant • Global Depth Offset Scale • Global Depth Offset Clamp
	7:0	**DWord Length** Default Value: 03h Excludes DWord (0,1) Format: =n Total Length - 2
1..4	127:0	**Raster State Body** Format: 3DSTATE_RASTER_BODY

3DSTATE_SAMPLE_MASK

3DSTATE_SAMPLE_MASK

Source:	RenderCS
Length Bias:	2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 18h 3DSTATE_SAMPLE_MASK Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**Dword Length** Default Value: 0h Excludes Dword (0,1) Format: =n
1	31:0	**Sample Mask State Body** Format: 3DSTATE_SAMPLE_MASK_BODY

3DSTATE_SAMPLE_PATTERN

3DSTATE_SAMPLE_PATTERN

Source:	RenderCS
Length Bias:	2

The 3DSTATE_SAMPLE_PATTERN command is used to specify the sample offsets for all multisample sample modes. The set of offset used will be selected based on the multisample mode. This is non-pipelined state.

Programming Notes
When programming the sample offsets (for NUMSAMPLES_4 or _8 and MSRASTMODE_xxx_PATTERN), the order of the samples 0 to 3 (or 7 for 8X, or 15 for 16X) must have monotonically increasing distance from the pixel center. This is required to get the correct centroid computation in the device.
Restriction : When executed in the POCS command stream, this command programs the multisample pattern state for the CLR and SFR stage of the POCS pipeline

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Ch 3DSTATE_SAMPLE_PATTERN Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 7 Excludes Dword (0,1) Format: =n
1	31:28	**16x Sample3 X Offset** Format: U0.4 Subpixel X offset of Sample 3 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	27:24	**16x Sample3 Y Offset** Format: U0.4 Subpixel Y offset of Sample 3 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	23:20	**16x Sample2 X Offset** Format: U0.4 Subpixel X offset of Sample 2 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	19:16	**16x Sample2 Y Offset** Format: U0.4 Subpixel Y offset of Sample 2 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES_4 , _8 or _16. Range: [0,0.9375]
	15:12	**16x Sample1 X Offset** Format: U0.4 Subpixel X offset of Sample 1 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES_2, _4 , _8 or _16. Range: [0,0.9375]
	11:8	**16x Sample1 Y Offset** Format: U0.4 Subpixel Y offset of Sample 1 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	7:4	**16x Sample0 X Offset** Format: U0.4 Subpixel X offset of Sample 0 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	3:0	**16x Sample0 Y Offset** Format: U0.4 Subpixel Y offset of Sample 0 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN		
2	31:28	**16x Sample7 X Offset** Format: U0.4 Subpixel X offset of Sample 7 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	27:24	**16x Sample7 Y Offset** Format: U0.4 Subpixel Y offset of Sample 7 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	23:20	**16x Sample6 X Offset** Format: U0.4 Subpixel X offset of Sample 6 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES__8 or _16. Range: [0,0.9375]
	19:16	**16x Sample6 Y Offset** Format: U0.4 Subpixel Y offset of Sample 6 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	15:12	**16x Sample5 X Offset** Format: U0.4 Subpixel X offset of Sample 5 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	11:8	**16x Sample5 Y Offset** Format: U0.4 Subpixel Y offset of Sample 5 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES__8 or _16. Range: [0,0.9375]
	7:4	**16x Sample4 X Offset** Format: U0.4 Subpixel X offset of Sample 4 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES__8 or _16. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	3:0	**16x Sample4 Y Offset** Format: U0.4 Subpixel Y offset of Sample 4 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
3	31:28	**16x Sample11 X Offset** Format: U0.4 Subpixel X offset of Sample 11 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	27:24	**16x Sample11 Y Offset** Format: U0.4 Subpixel Y offset of Sample 11 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES _16. Range: [0,0.9375]
	23:20	**16x Sample10 X Offset** Format: U0.4 Subpixel X offset of Sample 10 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	19:16	**16x Sample10 Y Offset** Format: U0.4 Subpixel Y offset of Sample 10 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES _16 Range: [0,0.9375]
	15:12	**16x Sample9 X Offset** Format: U0.4 Subpixel X offset of Sample 9 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	11:8	**16x Sample9 Y Offset** Format: U0.4 Subpixel Y offset of Sample 9 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN		
	7:4	**16x Sample8 X Offset** Format: U0.4 Subpixel X offset of Sample 8 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	3:0	**16x Sample8 Y Offset** Format: U0.4 Subpixel Y offset of Sample 8 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
4	31:28	**16x Sample15 X Offset** Format: U0.4 Subpixel X offset of Sample 15 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	27:24	**16x Sample15 Y Offset** Format: U0.4 Subpixel Y offset of Sample 15 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	23:20	**16x Sample14 X Offset** Format: U0.4 Subpixel X offset of Sample 14 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES _16. Range: [0,0.9375]
	19:16	**16x Sample14 Y Offset** Format: U0.4 Subpixel Y offset of Sample 14 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES _16 Range: [0,0.9375]
	15:12	**16x Sample13 X Offset** Format: U0.4 Subpixel X offset of Sample 13 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN		
	11:8	**16x Sample13 Y Offset** Format: U0.4 Subpixel Y offset of Sample 13 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
	7:4	**16x Sample12 X Offset** Format: U0.4 Subpixel X offset of Sample 12 relative to the UL pixel origin. Valid only when NUMRASTSAMPLES_16. Range: [0,0.9375]
	3:0	**16x Sample12 Y Offset** Format: U0.4 Subpixel Y offset of Sample 12 relative to the UL pixel origin for 16x mode. Range: [0,0.9375]
5	31:28	**8x Sample7 X Offset** Format: U0.4 Subpixel X offset of Sample 7 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	27:24	**8x Sample7 Y Offset** Format: U0.4 Subpixel Y offset of Sample 7 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	23:20	**8x Sample6 X Offset** Format: U0.4 Subpixel X offset of Sample 6 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	19:16	**8x Sample6 Y Offset** Format: U0.4 Subpixel Y offset of Sample 6 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	15:12	**8x Sample5 X Offset** Format: U0.4 Subpixel X offset of Sample 5 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	11:8	**8x Sample5 Y Offset** Format: U0.4 Subpixel Y offset of Sample 5 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	7:4	**8x Sample4 X Offset** Format: U0.4 Subpixel X offset of Sample 4 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	3:0	**8x Sample4 Y Offset** Format: U0.4 Subpixel Y offset of Sample 4 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
6	31:28	**8x Sample3 X Offset** Format: U0.4 Subpixel X offset of Sample 3 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	27:24	**8x Sample3 Y Offset** Format: U0.4 Subpixel Y offset of Sample 3 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	23:20	**8x Sample2 X Offset** Format: U0.4 Subpixel X offset of Sample 2 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	19:16	**8x Sample2 Y Offset** Format: U0.4 Subpixel Y offset of Sample 2 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	15:12	**8x Sample1 X Offset** Format: U0.4 Subpixel X offset of Sample 1 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	11:8	**8x Sample1 Y Offset** Format: U0.4 Subpixel Y offset of Sample 1 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	7:4	**8x Sample0 X Offset** Format: U0.4 Subpixel X offset of Sample 0 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	3:0	**8x Sample0 Y Offset** Format: U0.4 Subpixel Y offset of Sample 0 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
7	31:28	**4x Sample3 X Offset** Format: U0.4 Subpixel X offset of Sample 3 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	27:24	**4x Sample3 Y Offset** Format: U0.4 Subpixel Y offset of Sample 3 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN		
	23:20	**4x Sample2 X Offset** Format: U0.4 Subpixel X offset of Sample 2 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	19:16	**4x Sample2 Y Offset** Format: U0.4 Subpixel Y offset of Sample 2 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	15:12	**4x Sample1 X Offset** Format: U0.4 Subpixel X offset of Sample 1 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	11:8	**4x Sample1 Y Offset** Format: U0.4 Subpixel Y offset of Sample 1 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	7:4	**4x Sample0 X Offset** Format: U0.4 Subpixel X offset of Sample 0 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	3:0	**4x Sample0 Y Offset** Format: U0.4 Subpixel Y offset of Sample 0 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
8	31:24	**Reserved** Access: RO Format: MBZ
	23:20	**1x Sample0 X Offset** Format: U0.4 Subpixel X offset of Sample 0 relative to the UL pixel origin for 1x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	19:16	**1x Sample0 Y Offset** Format: U0.4 Subpixel Y offset of Sample 0 relative to the UL pixel origin for 1x mode. Range: [0,0.9375]
	15:12	**2x Sample1 X Offset** Format: U0.4 Subpixel X offset of Sample 1 relative to the UL pixel origin for 2x mode. Range: [0,0.9375]
	11:8	**2x Sample1 Y Offset** Format: U0.4 Subpixel Y offset of Sample 1 relative to the UL pixel origin for 2x mode. Range: [0,0.9375]
	7:4	**2x Sample0 X Offset** Format: U0.4 Subpixel X offset of Sample 0 relative to the UL pixel origin for 2x mode. Range: [0,0.9375]
	3:0	**2x Sample0 Y Offset** Format: U0.4 Subpixel Y offset of Sample 0 relative to the UL pixel origin for 2x mode. Range: [0,0.9375]

3DSTATE_SAMPLER_STATE_POINTERS_DS

3DSTATE_SAMPLER_STATE_POINTERS_DS

Source:	RenderCS
Length Bias:	2

The 3DSTATE_SAMPLER_STATE_POINTERS_DS command is used to define the location of DS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 2Dh 3DSTATE_SAMPLER_STATE_POINTERS_DS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Sampler State Pointers State Body** Format: **3DSTATE_SAMPLER_STATE_POINTERS_BODY**

3DSTATE_SAMPLER_STATE_POINTERS_GS

3DSTATE_SAMPLER_STATE_POINTERS_GS

Source:	RenderCS
Length Bias:	2

The 3DSTATE_SAMPLER_STATE_POINTERS_GS command is used to define the location of GS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 2Eh 3DSTATE_SAMPLER_STATE_POINTERS_GS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Sampler State Pointers State Body** Format: 3DSTATE_SAMPLER_STATE_POINTERS_BODY

3DSTATE_SAMPLER_STATE_POINTERS_HS

3DSTATE_SAMPLER_STATE_POINTERS_HS

Source:	RenderCS
Length Bias:	2

The 3DSTATE_SAMPLER_STATE_POINTERS_HS command is used to define the location of HS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 2Ch 3DSTATE_SAMPLER_STATE_POINTERS_HS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Sampler State Pointers State Body** Format: 3DSTATE_SAMPLER_STATE_POINTERS_BODY

3DSTATE_SAMPLER_STATE_POINTERS_PS

3DSTATE_SAMPLER_STATE_POINTERS_PS

Source:	RenderCS
Length Bias:	2

The 3DSTATE_SAMPLER_STATE_POINTERS_PS command is used to define the location of PS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.

DWord	Bit	Description	
0	31:29	**Command Type**	
		Default Value:	3h GFXPIPE
		Format:	OpCode
	28:27	**Command SubType**	
		Default Value:	3h GFXPIPE_3D
		Format:	OpCode
	26:24	**3D Command Opcode**	
		Default Value:	0h 3DSTATE_PIPELINED
		Format:	OpCode
	23:16	**3D Command Sub Opcode**	
		Default Value:	2Fh 3DSTATE_SAMPLER_STATE_POINTERS_PS
		Format:	OpCode
	15	**Reserved**	
		Access:	RO
		Format:	MBZ
	14:8	**Reserved**	
		Access:	RO
		Format:	MBZ
	7:0	**DWord Length**	
		Default Value:	0h DWORD_COUNT_n
		Format:	=n
1	31:0	**Sampler State Pointers State Body**	
		Format:	**3DSTATE_SAMPLER_STATE_POINTERS_BODY**

3DSTATE_SAMPLER_STATE_POINTERS_VS

3DSTATE_SAMPLER_STATE_POINTERS_VS

Source:	RenderCS
Length Bias:	2

The 3DSTATE_SAMPLER_STATE_POINTERS_VS command is used to define the location of VS SAMPLER_STATE table. Only some of the fixed functions utilize sampler state tables.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 2Bh 3DSTATE_SAMPLER_STATE_POINTERS_VS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Sampler State Pointers State Body** Format: 3DSTATE_SAMPLER_STATE_POINTERS_BODY

3DSTATE_SBE_MESH

3DSTATE_SBE_MESH		
Source:	RenderCS	
Length Bias:	2	
The 3DSTATE_SBE_MESH command is used to provide SBE with states required to process geometry generated by the Mesh Shader stage.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 82h 3DSTATE_SBE_MESH Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**SBE Mesh State Body** Format: 3DSTATE_SBE_MESH_BODY

3DSTATE_SBE

3DSTATE_SBE

Source:	RenderCS
Length Bias:	2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Fh 3DSTATE_SBE Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 04h Excludes DWord (0,1) Format: =n Total Length - 2
1..5	159:0	**SBE State Body** Format: 3DSTATE_SBE_BODY

3DSTATE_SBE_SWIZ

3DSTATE_SBE_SWIZ

Source:	RenderCS
Length Bias:	2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 51h 3DSTATE_SBE_SWIZ Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 9h Excludes DWord (0,1) Format: =n Total Length - 2
1..10	319:0	**SBE SWIZ State Body** Format: 3DSTATE_SBE_SWIZ_BODY

3DSTATE_SCISSOR_STATE_POINTERS

3DSTATE_SCISSOR_STATE_POINTERS

Source:	RenderCS
Length Bias:	2

The 3DSTATE_SCISSOR_STATE_POINTERS command is used to define the location of the indirect SCISSOR_RECT state.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Fh 3DSTATE_SCISSOR_STATE_POINTERS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Scissor State Pointers Body** Format: 3DSTATE_SCISSOR_STATE_POINTERS_BODY

3DSTATE_SF

3DSTATE_SF		
Source:	RenderCS	
Length Bias:	2	

Restriction
When executed in the POCS command stream, this command programs the state for the SFR stage of the POCS pipeline.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 13h 3DSTATE_SF Format: OpCode
	15:11	**Reserved** Access: RO Format: MBZ
	10	**SF State Modify Disable** Format: Disable When this bit is set, the following fields will be ignored: • Legacy Global Depth Bias Enable • Statistics Enable • Viewport Transform Enable • Fast Scissor Clip Disable • Line End Cap Antialiasing Region Width • Zero Pixel Triangle Filter Disable • 2x2 Pixel Triangle Filter Disable • Last Pixel Enable • Triangle Strip/List Provoking Vertex Select • Line Strip/List Provoking Vertex Select • Triangle Fan Provoking Vertex Select

3DSTATE_SF

		• AA Line Distance Mode • Smooth Point Enable • Vertex Sub Pixel Precision Select • Point Width Source
	9	**Line Width Modify Disable** Format: | Disable When this bit is set, the following fields will be ignored: • Line Width
	8	**Point Width Modify Disable** Format: | Disable When this bit is set, the following fields will be ignored: • Point Width
	7:0	**DWord Length** Default Value: | 2h Excludes DWord (0,1) Format: | =n Total Length - 2
1..3	95:0	**SF State Body** Format: | 3DSTATE_SF_BODY

3DSTATE_SLICE_TABLE_STATE_POINTERS

3DSTATE_SLICE_TABLE_STATE_POINTERS

Source: RenderCS

Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: Opcode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 20h 3DSTATE_SLICE_TABLE_STATE_POINTERS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n
1	31:6	**Slice Hash Table State Pointer** Format: DynamicStateOffset[31:6]SLICE_HASH_TABLE Specifies the 64-byte aligned offset of the SLICE_HASH_TABLE. This offset is relative to the **Dynamic State Base Address**.
	5:1	**Reserved** Access: RO Format: MBZ
	0	**Slice Hash State Pointer Valid** Format: Enable This bit, if set, indicates that the SLICE_HASH_TABLE pointer has changed and new state needs to be fetched. Value: 0h – Name: Disable Value: 1h – Name: Enable

3DSTATE_SO_BUFFER_INDEX_0

3DSTATE_SO_BUFFER_INDEX_0		
Source: RenderCS Length Bias: 2		
This command is to program the SO buffer addresses and Attributes for Index 0.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 60h 3DSTATE_SO_BUFFER_INDEX_0 Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 6h Excludes DWord (0,1) Format: =n Total Length - 2
1..7	223:0	**SO Buffer Index State Body** Format: 3DSTATE_SO_BUFFER_INDEX_BODY

3DSTATE_SO_BUFFER_INDEX_1

3DSTATE_SO_BUFFER_INDEX_1		
Source:	RenderCS	
Length Bias:	2	
This command is to program the SO buffer addresses and Attributes for Index 1.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 61h 3DSTATE_SO_BUFFER_INDEX_1 Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 6h Excludes DWord (0,1) Format: =n Total Length - 2
1..7	223:0	**SO Buffer Index State Body** Format: 3DSTATE_SO_BUFFER_INDEX_BODY

3DSTATE_SO_BUFFER_INDEX_2

3DSTATE_SO_BUFFER_INDEX_2		
Source: RenderCS Length Bias: 2		
This command is to program the SO buffer addresses and Attributes for Index 2.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 62h 3DSTATE_SO_BUFFER_INDEX_2 Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 6h Excludes DWord (0,1) Format: =n Total Length - 2
1..7	223:0	**SO Buffer Index State Body** Format: 3DSTATE_SO_BUFFER_INDEX_BODY

3DSTATE_SO_BUFFER_INDEX_3

3DSTATE_SO_BUFFER_INDEX_3

Source:	RenderCS
Length Bias:	2

This command is to program the SO buffer addresses and Attributes for Index 3.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 63h 3DSTATE_SO_BUFFER_INDEX_3 Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 6h Excludes DWord (0,1) Format: =n Total Length - 2
1..7	223:0	**SO Buffer Index State Body** Format: 3DSTATE_SO_BUFFER_INDEX_BODY

3DSTATE_SO_BUFFER

3DSTATE_SO_BUFFER

Source:	RenderCS
Length Bias:	2

Programming Notes
Foreach SO Buffer, the 3DSTATE_SO_BUFFER must only be sent once prior to each 3DPRIMITIVE command.

Workaround
This command must be followed by a PIPE_CONTROL with CS Stall bit set.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 18h 3DSTATE_SO_BUFFER Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 6h Excludes DWord (0,1) Format: =n Total Length - 2
1	31	**SO Buffer Enable** Format: Enable If set, stream output to SO Buffer is enabled, , if 3DSTATE_STREAMOUT::SO Function ENABLE is also enabled. If clear, the SO Buffer is considered "not bound" and effectively treated as a zero-length buffer for the purposes of SO output and overflow detection. If an enabled stream's Stream to Buffer Selects includes this buffer it is by definition an overflow condition. That stream will cause no writes to occur, and only SO_PRIM_STORAGE_NEEDED[<stream>] will increment.

3DSTATE_SO_BUFFER		
	30:29	**SO Buffer Index** Format: | U2 Specifies which of the four SO Buffers is being defined.
	28:22	**SO Buffer Object Control State** Format: | MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for the SO buffer.
	21	**Stream Offset Write Enable** Format: | Enable When set, this field allows the hardware to write SO_WRITE_OFFSET[Buffer#] as specified in the Stream Offset field. **Programming Notes** The field operates irrespective of whether SO Buffer Enable is set or clear.
	20	**Stream Output Buffer Offset Address Enable** Format: | Enable When set, this field allows the hardware to read/write the stream output buffer offset as specified in the "Stream Output Buffer Offset Address" field. **Programming Notes** The field operates irrespective of whether SO Buffer Enable is set or clear.
	19:0	**Reserved** Access: | RO Format: | MBZ
2..3	63:2	**Surface Base Address** Format: | VIRTUAL_ADDR[63:2] This field specifies the starting address of the buffer in Graphics Memory.
	1:0	**Reserved** Access: | RO Format: | MBZ
4	31:30	**Reserved** Access: | RO Format: | MBZ
	29:0	**Surface Size** Format: | U30-1 This field specifies the size of buffer in number DWords minus 1 of the buffer in Graphics Memory.
5..6	63:2	**Stream Output Buffer Offset Address** Format: | VIRTUAL_ADDR[63:2] This field specifies the starting address of the buffer in Graphics Memory where the Stream Output Buffer Offset is stored when all the data has been written. It is also used to fetch the stream Output buffer Offset when needed.

<table>
<tr><th colspan="3">3DSTATE_SO_BUFFER</th></tr>
<tr><td></td><td>1:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>7</td><td>31:0</td><td><b>Stream Offset</b>
This field specifies the Offset in stream output buffer to start at, or whether to append to the end of an existing buffer. The Offset must be DWORD aligned. If Stream Offset is equal to 0xFFFFFFFF then load the value at the Stream Output Buffer Offset address into SO_WRITE_OFFSET[Buffer#]. Otherwise, SO_WRITE_OFFSET[Buffer#] = Stream Offset.</td></tr>
</table>

3DSTATE_SO_DECL_LIST

<table>
<tr><th colspan="3">3DSTATE_SO_DECL_LIST</th></tr>
<tr><td colspan="3">Source: RenderCS
Length Bias: 2</td></tr>
<tr><td colspan="3"><table><tr><th>Workaround</th></tr><tr><td>This command must be followed by a PIPE_CONTROL with CS Stall bit set.,</td></tr></table></td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h GFXPIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Command SubType</b><table><tr><td>Default Value:</td><td>3h GFXPIPE_3D</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><b>3D Command Opcode</b><table><tr><td>Default Value:</td><td>1h 3DSTATE_NONPIPELINED</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>3D Command Sub Opcode</b><table><tr><td>Default Value:</td><td>17h 3DSTATE_SO_DECL_LIST</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:9</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>8:0</td><td><b>DWord Length</b><table><tr><td>Format:</td><td>=n</td></tr></table><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>[1,257]</td><td>Excludes DWORD (0,1) 0-128 Entries</td><td>Value = 2 * (# of SO_DECL quads) + 1</td></tr></table></td></tr>
<tr><td>1</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Stream to Buffer Selects [3]</b><table><tr><td>Format:</td><td>U4</td></tr></table>Identifies to which SO Buffers stream 3 outputs. See Stream To Buffer Selects [0] field description.<table><tr><th>Value</th><th>Name</th></tr><tr><td>1xxxb</td><td>SO Buffer 3</td></tr><tr><td>x1xxb</td><td>SO Buffer 2</td></tr></table></td></tr>
</table>

3DSTATE_SO_DECL_LIST

xx1xb	SO Buffer 1
xxx1b	SO Buffer 0

11:8 **Stream to Buffer Selects [2]**

Format:	U4

Identifies to which SO Buffers stream 2 outputs. See Stream To Buffer Selects [0] field description.

Value	Name
1xxxb	SO Buffer 3
x1xxb	SO Buffer 2
xx1xb	SO Buffer 1
xxx1b	SO Buffer 0

7:4 **Stream to Buffer Selects [1]**

Format:	U4

Identifies to which SO Buffers stream 1 outputs. See Stream To Buffer Selects [0] field description.

Value	Name
1xxxb	SO Buffer 3
x1xxb	SO Buffer 2
xx1xb	SO Buffer 1
xxx1b	SO Buffer 0

3:0 **Stream to Buffer Selects [0]**

Format:	U4

Identifies to which SO Buffers stream 0 outputs (irrespective of whether those buffers are enabled via 3DSTATE_STREAMOUT). Software is required to scan the SO_DECL list in order to provide this summary information. Note: For "inactive" streams, software must program this field to all zero (no buffers written to) and the corresponding Num Entries field to zero (no valid SO_DECLs).

Value	Name
1xxxb	SO Buffer 3
x1xxb	SO Buffer 2
xx1xb	SO Buffer 1
xxx1b	SO Buffer 0

2 31:24 **Num Entries [3]**

Format:	U8

Specifies the number of valid SO_DECL entries for Stream 3. (See notes in Num Entries [0] field description).

Value	Name
[0,128]	entries

<table>
<tr><th colspan="3">3DSTATE_SO_DECL_LIST</th></tr>
<tr><td></td><td>23:16</td><td><b>Num Entries [2]</b>
<table><tr><td>Format:</td><td>U8</td></tr></table>
Specifies the number of valid SO_DECL entries for Stream 2. (See notes in Num Entries [0] field description).
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,128]</td><td>entries</td></tr></table></td></tr>
<tr><td></td><td>15:8</td><td><b>Num Entries [1]</b>
<table><tr><td>Format:</td><td>U8</td></tr></table>
Specifies the number of valid SO_DECL entries for Stream 1. (See notes in Num Entries [0] field description).
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,128]</td><td>entries</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>Num Entries [0]</b>
<table><tr><td>Format:</td><td>U8</td></tr></table>
Specifies the number of valid SO_DECL entries for Stream 0. Note that the SO_DECLs are programmed in groups of four (one SO_DECL for each of the four streams). Therefore the number of 2-DWord groups of SO_DECLs supplied in this command is derived from the stream(s) with the most valid SO_DECLs. The NumEntries value specific to each stream will indicate how many SO_DECLS are valid for that particular stream. Any trailing invalid SO_DECLs supplied for streams with fewer valid SO_DECLs will be ignored. It is legal to specify Num Entries = 0 for all four streams simultaneously. In this case there will be no SO_DECLs included in the command (only DW 0-2). Note that all Stream to Buffer Selects bits must be zero in this case (as no streams produce output).
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,128]</td><td>entries</td></tr></table></td></tr>
<tr><td>3..n</td><td>63:0</td><td><b>Entry</b>
<table><tr><td>Format:</td><td>SO_DECL_ENTRY</td></tr></table></td></tr>
</table>

3DSTATE_STENCIL_BUFFER

3DSTATE_STENCIL_BUFFER

Source:	RenderCS
Length Bias:	2

The stencil buffer surface state is delivered as a pipelined state packet. However, the state change pipelining isn't completely transparent (see restriction below).

WM HW will internally manage the draining pipe and flushing of the caches when this command is issued. The PIPE_CONTROL restrictions are removed.

Programming Notes
If the Stencil surface is not present, SW must set the Surface Type field to SURFTYPE_NULL.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 6h 3DSTATE_STENCIL_BUFFER Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 6h Excludes Dword (0,1) Format: =n Excludes DWord(0,1)
1..7	223:0	**Stencil Buffer State Body** Format: 3DSTATE_STENCIL_BUFFER_BODY

3DSTATE_STREAMOUT

3DSTATE_STREAMOUT

Source:	RenderCS
Length Bias:	2

This command contains pipelined state required by the SOL unit.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Eh 3DSTATE_STREAMOUT Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 3h Excludes DWord (0,1) Format: =n
1..4	127:0	**Streamout State Body** Format: 3DSTATE_STREAMOUT_BODY

3DSTATE_SUBSLICE_HASH_TABLE

3DSTATE_SUBSLICE_HASH_TABLE

Source:	RenderCS
Length Bias:	2

Programmable Dual SubSlice hashing control and tables. First DW indicates the mode how the last 4DW are configured and selects the programmable dual subslice hashing mode for each slice.

Programming Notes
All slice references are physical slice. Instruction must be programmed based on which slices and subslices are enabled.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table><tr><td>Default Value:</td><td>3h GFXPIPE</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Command SubType</b>
<table><tr><td>Default Value:</td><td>3h GFXPIPE_3D</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><b>3D Command Opcode</b>
<table><tr><td>Default Value:</td><td>1h 3DSTATE_NONPIPELINED</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>3D Command Sub Opcode</b>
<table><tr><td>Default Value:</td><td>1Fh 3DSTATE_SUBSLICE_HASH_TABLE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:8</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table><tr><td>Format:</td><td>=n</td></tr></table>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>Ch</td><td>Excludes DWord (0,1) <b>[Default]</b></td></tr></table></td></tr>
<tr><td>1</td><td>31:30</td><td><b>TableMode</b>
<table><tr><td>Format:</td><td>U2</td></tr></table>
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Single table <b>[Default]</b></td><td>DW2-5 is Table[0] - a single 2-way 128 entry [Y][X] table.</td></tr><tr><td>1h</td><td>Dual tables</td><td>DW2-3 is 'Table[0]' and is 2-way 64 entry [Y][X] table.
DW4-5 is 'Table[1]' and is 2-way 64 entry [Y][X] table.</td></tr></table></td></tr>
</table>

3DSTATE_SUBSLICE_HASH_TABLE

	29:16	**Reserved** Access: RO Format: MBZ
	15:0	**SliceHashCtrl** PerSlice[7:0]SliceHashControl
2..5	127:0	**64 Entry 2-way Tables** Exists If: Instruction[3DSTATE_SUBSLICE_HASH_TABLE][TableMode]==Dual Format: U1[8][8][2] **Description** 2-way pixel hashing tables. Tables are 64-entries:8X,8Y in [Y][X] format. Each entry is a single bit that indicates which sub-slice hardware block the indicated xy pixel block is mapped. pixelhash_id maps to color-pipe. A value of 0 indicates the larger color-pipe, or first enabled color-pipe if both enabled color-pipes are balanced
	127:0	**128 Entry 2-way Table** Exists If: Instruction[3DSTATE_SUBSLICE_HASH_TABLE][TableMode]==Single Format: U1[8][16] 2-way pixel hashing table. Table is 128-entries:16X,8Y in [Y][X] format. Each entry is a single bit that indicates which sub-slice hardware block the indicated xy pixel block is mapped.
6..13	255:0	**64 Entry 3-way Tables** Exists If: Instruction[3DSTATE_SUBSLICE_HASH_TABLE][TableMode]==Dual Format: U2[8][8][2] **Description** 3-way or 4-way pixel hashing tables. Tables are 64-entries:8X,8Y in [Y][X] format. Each entry is two bits that indicates which sub-slice hardware block the indicated xy pixel block is mapped. pixelhash_id maps to color-pipe. A value of 0 indicates the largest color-pipe, or first enabled color-pipe if all enabled color-pipes are balanced. A value of 2 indicates the smallest color-pipe, or last enabled color-pipe if all enabled color-pipes are balanced.
	255:0	**128 Entry 3-way Table** Exists If: Instruction[3DSTATE_SUBSLICE_HASH_TABLE][TableMode]==Single Format: U2[16][8] **Description** 3-way or 4-way pixel hashing table. Table is 128-entries:16X,8Y in [Y][X] format. Each entry is two bits that indicates which sub-slice hardware block the indicated xy pixel block is mapped. pixelhash_id maps to color-pipe. A value of 0 indicates the largest color-pipe, or first enabled color-pipe if all enabled color-pipes are balanced. A value of 2 indicates the smallest color-pipe, or last enabled color-pipe if all enabled color-pipes are balanced.

3DSTATE_TASK_CONTROL

3DSTATE_TASK_CONTROL - 3DSTATE_TASK_CONTROL

Source:	BSpec
Length Bias:	2

Specifies low-frequency control states for the Task Shader stage.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 7Ch 3DSTATE_TASK_CONTROL Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n
1..2	63:0	**Task Shader Control Body** Format: 3DSTATE_TASK_CONTROL_BODY

3DSTATE_TASK_REDISTRIB

3DSTATE_TASK_REDISTRIB - 3DSTATE_TASK_REDISTRIB

Source:	BSpec
Length Bias:	2

Specifies states that control the redistribution of TaskShader-generated meshes across lower geometry pipelines.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 79h 3DSTATE_TASK_REDISTRIB Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n
1	31:0	**Task Shader Redistrib Body** Format: 3DSTATE_TASK_REDISTRIB_BODY

3DSTATE_TASK_SHADER_DATA

3DSTATE_TASK_SHADER_DATA - 3DSTATE_TASK_SHADER_DATA

Source:	BSpec
Length Bias:	2

Specifies data-input states for the Task Shader stage.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 7Eh 3DSTATE_TASK_SHADER_DATA Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 8h Excludes DWord (0,1) Format: =n
1..9	287:0	**Task Shader Data Body** Format: 3DSTATE_TASK_SHADER_DATA_BODY

3DSTATE_TASK_SHADER

3DSTATE_TASK_SHADER - 3DSTATE_TASK_SHADER		
Source:	BSpec	
Length Bias:	2	
Specifies shader-related states for the Task Shader stage.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 7Dh 3DSTATE_TASK_SHADER Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 5h Excludes DWord (0,1) Format: =n
1..6	191:0	**Task Shader Body** Format: 3DSTATE_TASK_SHADER_BODY

3DSTATE_TBIMR_TILE_PASS_INFO

3DSTATE_TBIMR_TILE_PASS_INFO

Source:	RenderCS
Length Bias:	2

The 3DSTATE_PTBR_TBIMR_PASS_INFO command is used to define the required parameters for a Tile Pass in TBIMR mode.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 6Eh 3DSTATE_TBIMR_TILE_PASS_INFO Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 2h Not include DW0 and DW1 Format: =n
1..3	95:0	**TBIMR Tile Pass Info State Body** Format: 3DSTATE_TBIMR_TILE_PASS_INFO_BODY

3DSTATE_TE

3DSTATE_TE

Source:	RenderCS
Length Bias:	2

The state used by TE is defined with this inline state packet.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Ch 3DSTATE_TE Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Format: =n Value | Name 3h | Excludes DWord (0,1) **[Default]**
1..4	127:0	**TE State Body** Format: 3DSTATE_TE_BODY

3DSTATE_URB_ALLOC_DS

3DSTATE_URB_ALLOC_DS

Source:	RenderCS
Length Bias:	2

When executed from the RCS command stream, this command provides the state variables associated with the URB region used by the DS pipeline stage.

Separate state variables are provided to define the URB region for Slice0 as well as additional URB space for additional slices (if any are enabled). Hardware will use those values to automatically compute the URB allocation within the total URB space based on the number of enabled slices. Software shall ensure that the values programmed do not exceed the URB capacity of a slice. Refer to the L3 allocation and programming guide for valid URB configurations.

Programming Notes
SW shall ensure that the ordering of URB allocations is consistent between the Slice0 and SliceN state settings across all FF stage URB allocations.
SW shall issue allocation state for all FF stages, i.e., 3DSTATE_URB_ALLOC_VS, 3DSTATE_URB_ALLOC_HS, 3DSTATE_URB_ALLOC_DS, and 3DSTATE_URB_ALLOC_GS commands.
If SW supports use of the Task or Mesh shader stages, when specifying URB allocation state for the RCS 3D pipe, it shall also issue allocation state for the Task and Mesh stages, i.e., 3DSTATE_URB_ALLOC_TASK, 3DSTATE_URB_ALLOC_MESH commands.
If Domain Shader Thread Dispatch is Enabled then the minimum number of handles that must be allocated is 50 URB entries.
SW shall ensure that the URB region specified by this command does not overlap with the push constant allocation in the URB, as defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 5Ah 3DSTATE_URB_ALLOC_DS Format: OpCode

3DSTATE_URB_ALLOC_DS

	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 1h DWORD_COUNT_n Format: =n
1..2	63:0	**URB Alloc DS State Body** Format: 3DSTATE_URB_ALLOC_DS_BODY

3DSTATE_URB_ALLOC_GS

3DSTATE_URB_ALLOC_GS

Source:	RenderCS
Length Bias:	2

When executed from the RCS command stream, this command provides the state variables associated with the URB region used by the GS pipeline stage.

Separate state variables are provided to define the URB region for Slice0 as well as additional URB space for additional slices (if any are enabled). Software shall ensure that the values programmed do not exceed the URB capacity of a slice. Refer to the L3 allocation and programming guide for valid URB configurations.

Programming Notes
SW shall ensure that the ordering of URB allocations is consistent between the Slice0 and SliceN state settings across all FF stage URB allocations.
SW shall issue allocation state for all FF stages, i.e., 3DSTATE_URB_ALLOC_VS, 3DSTATE_URB_ALLOC_HS, 3DSTATE_URB_ALLOC_DS, and 3DSTATE_URB_ALLOC_GS commands.
SW shall ensure that the URB region specified by this command does not overlap with the push constant allocation in the URB, as defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 5Bh 3DSTATE_URB_ALLOC_GS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 1h DWORD_COUNT_n Format: =n

3DSTATE_URB_ALLOC_GS		
1..2	63:0	**URB Alloc GS State Body** Format: **3DSTATE_URB_ALLOC_GS_BODY**

3DSTATE_URB_ALLOC_HS

3DSTATE_URB_ALLOC_HS

Source:	RenderCS
Length Bias:	2

When executed from the RCS command stream, this command provides the state variables associated with the URB region used by the HS pipeline stage.

Separate state variables are provided to define the URB region for Slice0 as well as additional URB space for additional slices (if any are enabled). Hardware will use those values to automatically compute the URB allocation within the total URB space based on the number of enabled slices. Software shall ensure that the values programmed do not exceed the URB capacity of a slice. Refer to the L3 allocation and programming guide for valid URB configurations.

Programming Notes
SW shall ensure that the ordering of URB allocations is consistent between the Slice0 and SliceN state settings across all FF stage URB allocations.
SW shall issue allocation state for all FF stages, i.e., 3DSTATE_URB_ALLOC_VS, 3DSTATE_URB_ALLOC_HS, 3DSTATE_URB_ALLOC_DS, and 3DSTATE_URB_ALLOC_GS commands.
SW shall ensure that the URB region specified by this command does not overlap with the push constant allocation in the URB, as defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 59h 3DSTATE_URB_ALLOC_HS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ

3DSTATE_URB_ALLOC_HS		
	7:0	**DWord Length** Default Value: 1h DWORD_COUNT_n Format: =n
1..2	63:0	**URB Alloc HS State Body** Format: 3DSTATE_URB_ALLOC_HS_BODY

3DSTATE_URB_ALLOC_MESH

3DSTATE_URB_ALLOC_MESH

Source:	RenderCS
Length Bias:	2

When executed from the RCS command stream, this command provides the state variables associated with the URB region used by the MESH pipeline stage.

Software shall ensure that the values programmed do not exceed the URB capacity of a slice. Refer to the L3 allocation and programming guide for valid URB configurations.

Programming Notes

SW shall ensure that the URB region specified by this command does not overlap with the push constant allocation in the URB, as defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.

If SW supports use of the Task or Mesh shader stages, when specifying URB allocation state for the RCS 3D pipe, it shall also issue allocation state for the Task and Mesh stages, i.e., 3DSTATE_URB_ALLOC_TASK, 3DSTATE_URB_ALLOC_MESH commands.

When specifying URB allocation state for the Mesh or Task shader stages, SW shall also issue allocation state for the other RCS 3D pipeline stages, i.e., 3DSTATE_URB_ALLOC_VS, 3DSTATE_URB_ALLOC_HS, 3DSTATE_URB_ALLOC_DS and 3DSTATE_URB_ALLOC_GS commands.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 7Fh 3DSTATE_URB_ALLOC_MESH Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 1h DWORD_COUNT_n Format: =n

3DSTATE_URB_ALLOC_MESH		
1..2	63:0	**URB Alloc MESH State Body** Format: 3DSTATE_URB_ALLOC_MESH_BODY

3DSTATE_URB_ALLOC_TASK

3DSTATE_URB_ALLOC_TASK

Source:	RenderCS
Length Bias:	2

When executed from the RCS command stream, this command provides the state variables associated with the URB region used by the TASK pipeline stage.

Software shall ensure that the values programmed do not exceed the URB capacity of a slice. Refer to the L3 allocation and programming guide for valid URB configurations.

Programming Notes

SW shall ensure that the URB region specified by this command does not overlap with the push constant allocation in the URB, as defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.

When specifying URB allocation state for the Mesh or Task shader stages, SW shall also issue allocation state for the other RCS 3D pipeline stages, i.e., 3DSTATE_URB_ALLOC_VS, 3DSTATE_URB_ALLOC_HS, 3DSTATE_URB_ALLOC_DS and 3DSTATE_URB_ALLOC_GS commands.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 80h 3DSTATE_URB_ALLOC_TASK Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 1h DWORD_COUNT_n Format: =n
1..2	63:0	**URB Alloc TASK State Body** Format: 3DSTATE_URB_ALLOC_TASK_BODY

3DSTATE_URB_ALLOC_VS

3DSTATE_URB_ALLOC_VS

Source:	RenderCS, PositionCS
Length Bias:	2

When executed from the RCS command stream, this command provides the state variables associated with the URB region used by the VF and VS pipeline stages.

When executed from the POCS command stream, this command provides the state variables associated with the URB region used by the VFR and VSR pipeline stages. The POCS command stream will only execute 3DSTATE_URB_ALLOC_VS command with respect to URB programming for the POCS 3D pipeline. 3DSTATE_URB_ALLOC_HS, 3DSTATE_URB_ALLOC_DS and 3DSTATE_URB_ALLOC_GS commands are ignored by the POCS command stream.

Separate state variables are provided to define the URB region for Slice0 as well as additional URB space for additional slices (if any are enabled). Hardware will use those values to automatically compute the URB allocation within the total URB space based on the number of enabled slices. Software shall ensure that the values programmed do not exceed the URB capacity of a slice. Refer to the L3 allocation and programming guide for valid URB configurations.

Programming Notes

SW shall ensure that the ordering of URB allocations is consistent between the Slice0 and SliceN state settings across all FF stage URB allocations.

The VSR URB region shall never overlap any other URB region. As POCS and RCS command streams are not implicitly synchronized, if POCS is used SW shall reserve a region of the URB for use by VSR. Both pipelines must be flushed and synchronized before expanding the VSR URB region such that the new VSR URB region overlaps URB space previously used by the render pipeline, or the URB space to be used by the render pipeline overlaps the previous VSR URB region.

When specifying URB allocation state for the RCS 3D pipe, SW shall issue allocation state for all FF stages, i.e., 3DSTATE_URB_ALLOC_VS, 3DSTATE_URB_ALLOC_HS, 3DSTATE_URB_ALLOC_DS, and 3DSTATE_URB_ALLOC_GS commands.

When specifying URB allocation state for the RCS 3D pipe, SW shall also issue allocation state for the Task and Mesh stages, i.e., 3DSTATE_URB_ALLOC_TASK, 3DSTATE_URB_ALLOC_MESH commands.

SW shall ensure that the URB region specified by this command does not overlap with the push constant allocation in the URB, as defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode

3DSTATE_URB_ALLOC_VS

	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 58h 3DSTATE_URB_ALLOC_VS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 1h DWORD_COUNT_n Format: =n
1..2	63:0	**URB Alloc VS State Body** Format: 3DSTATE_URB_ALLOC_VS_BODY

3DSTATE_URB_DS

3DSTATE_URB_DS

Source:	RenderCS
Length Bias:	2

This command may not overlap with the push constants in the URB defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.

The URB Starting Address and Number of URB Entries fields shall be programmed as if there is only one slice enabled. When more than one slice is enabled, hardware will (a) recompute the actual URB Starting Address based on the number of enabled slices and (b) multiply the Number of URB Entries by the number of enabled slices. Software shall ensure that the values programmed do not exceed the URB capacity of a single slice. Refer to the L3 allocation and programming guide for valid URB configurations.

Programming Notes
When programming DS URB state for the RCS 3D pipe, 3DSTATE_URB_VS, 3DSTATE_URB_HS, and 3DSTATE_URB_GS must also be programmed in order for the programming of this state to be valid.
Please see **3DSTATE_URB_ALLOC_DS** for any new programming notes related to URB programming.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 32h 3DSTATE_URB_DS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**URB DS State Body** Format: **3DSTATE_URB_DS_BODY**

3DSTATE_URB_GS

3DSTATE_URB_GS

Source:	RenderCS
Length Bias:	2

This command may not overlap with the push constants in the URB defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.

The URB Starting Address and Number of URB Entries fields shall be programmed as if there is only one slice enabled. When more than one slice is enabled, hardware will (a) recompute the actual URB Starting Address based on the number of enabled slices and (b) multiply the Number of URB Entries by the number of enabled slices. Software shall ensure that the values programmed do not exceed the URB capacity of a single slice. Refer to the L3 allocation and programming guide for valid URB configurations

Programming Notes
When programming GS URB state for the RCS 3D pipe, 3DSTATE_URB_VS, 3DSTATE_URB_HS, and 3DSTATE_URB_DS must also be programmed in order for the programming of this state to be valid.
Please see **3DSTATE_URB_ALLOC_GS** for any new programming notes related to URB programming.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 33h 3DSTATE_URB_GS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**URB GS State Body** Format: **3DSTATE_URB_GS_BODY**

3DSTATE_URB_HS

3DSTATE_URB_HS

Source:	RenderCS
Length Bias:	2

This command may not overlap with the push constants in the URB defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.

The URB Starting Address and Number of URB Entries fields shall be programmed as if there is only one slice enabled. When more than one slice is enabled, hardware will (a) recompute the actual URB Starting Address based on the number of enabled slices and (b) multiply the Number of URB Entries by the number of enabled slices. Software shall ensure that the values programmed do not exceed the URB capacity of a single slice. Refer to the L3 allocation and programming guide for valid URB configurations

Programming Notes

When programming HS URB state for the RCS 3D pipe, 3DSTATE_URB_VS, 3DSTATE_URB_DS, and 3DSTATE_URB_GS must also be programmed in order for the programming of this state to be valid.

Please see **3DSTATE_URB_ALLOC_HS** for any new programming notes related to URB programming.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 31h 3DSTATE_URB_HS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**URB HS State Body** Format: **3DSTATE_URB_HS_BODY**

3DSTATE_URB_VS

3DSTATE_URB_VS

Source:	RenderCS, PositionCS
Length Bias:	2

Description
VS URB Entry Allocation Size equal to 4(5 512-bit URB rows) may cause performance to decrease due to banking in the URB. Element sizes of 16 to 20 should be programmed with six 512-bit URB rows.
This command may not overlap with the push constants in the URB defined by the 3DSTATE_PUSH_CONSTANT_ALLOC_VS, 3DSTATE_PUSH_CONSTANT_ALLOC_DS, 3DSTATE_PUSH_CONSTANT_ALLOC_HS, and 3DSTATE_PUSH_CONSTANT_ALLOC_GS commands.
The offset and size should be programmed as if there is only one slice enabled. Hardware will grow the size based on the slice configuration. Software shall ensure that the values programmed do not exceed the URB capacity of one slice. Refer to the L3 allocation and programming guide for valid URB configurations.

Programming Notes
When programming VS URB state for the RCS 3D pipe, 3DSTATE_URB_HS, 3DSTATE_URB_DS, and 3DSTATE_URB_GS must also be programmed in order for the programming of this state to be valid.
Please see **3DSTATE_URB_ALLOC_VS** for any new programming notes related to URB programming.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 30h 3DSTATE_URB_VS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n

3DSTATE_URB_VS		
1	31:0	**URB VS State Body** Format: 3DSTATE_URB_VS_BODY

3DSTATE_VERTEX_BUFFERS

3DSTATE_VERTEX_BUFFERS

Source:	RenderCS
Length Bias:	2

This command is used to specify VB state used by the VF function.

Can specify from 1 to 33 VBs.

The VertexBufferID field within a VERTEX_BUFFER_STATE structure indicates the specific VB. If a VB definition is not included in this command, its associated state is left unchanged and is available for use if previously defined.

Programming Notes

It is possible to have individual vertex elements sourced completely from generated ID values and therefore not require any vertex buffer accesses for that vertex element. In this case, VF function will simply ignore the VB state associated with that vertex element. If all enabled vertex elements have this characteristic, no VBs are required to process 3DPRIMITIVE commands. For example, this might arise when the user wants to perform all data lookups in the first shader, so only generated index values need to be passed down to it. In this extreme case, SW would not need to program any VB state, and therefore not need to issue any 3DSTATE_VERTEX_BUFFERS commands.

For any 3DSTATE_VERTEX_BUFFERS command, at least one VERTEX_BUFFER_STATE structure must be included.

VERTEX_BUFFER_STATE structures are 4 DWords for both VERTEXDATA buffers and INSTANCEDATA buffers.

Inclusion of partial VERTEX_BUFFER_STATE structures is UNDEFINED.

The order in which VBs are defined within this command can be arbitrary, though a vertex buffer must be defined only once in any given command (otherwise operation is UNDEFINED).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 03h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: Opcode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: Opcode
	23:16	**3D Command Sub Opcode** Default Value: 08h 3DSTATE_VERTEX_BUFFERS Format: Opcode
	15	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">3DSTATE_VERTEX_BUFFERS</th></tr>
<tr><td></td><td>14:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>n = 4b-1 (where b = # of buffer states included)<table><tr><th>Value</th><th>Name</th></tr><tr><td>3</td><td>DWORD_COUNT_n <b>[Default]</b></td></tr><tr><td>[3,131]</td><td>1-33 Buffers</td></tr></table></td></tr>
<tr><td>1..n</td><td>127:0</td><td><b>Vertex Buffer State</b><table><tr><td>Format:</td><td>VERTEX_BUFFER_STATE</td></tr></table></td></tr>
</table>

3DSTATE_VERTEX_ELEMENTS

3DSTATE_VERTEX_ELEMENTS

Source:	RenderCS
Length Bias:	2

This is a variable-length command used to specify the active vertex elements. Each VERTEX_ELEMENT_STATE structure contains a Valid bit which determines which elements are used. Any elements not programmed by this command are disabled.

Up to 34 elements.

Programming Notes
At least one VERTEX_ELEMENT_STATE structure must be included.
Inclusion of partial VERTEX_ELEMENT_STATE structures is UNDEFINED.
SW must ensure that at least one vertex element is defined prior to issuing a 3DPRIMTIVE command, or operation is UNDEFINED.
There are no 'holes' allowed in the destination vertex: NOSTORE components must be overwritten by subsequent components unless they are the trailing DWords of the vertex. Software must explicitly chose some value (probably 0) to be written into DWords that would otherwise be 'holes'.
Within a VERTEX_ELEMENT_STATE structure, if a Component Control field is set to something other than VFCOMP_STORE_SRC, no higher-numbered Component Control fields may be set to VFCOMP_STORE_SRC. In other words, only trailing components can be set to something other than VFCOMP_STORE_SRC.
See additional restrictions listed in the command fields and VERTEX_ELEMENT_STATE description.
Element[0] must be valid.
All elements must be valid from Element[0] to the last valid element. (E.g.. if Element[2] is valid then Element[1] and Element[0] must also be valid).
The pitch between elements packed in the URB will always be 128 bits.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 03h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 3h 3D Format: Opcode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: Opcode
	23:16	**3D Command Sub Opcode** Default Value: 09h 3DSTATE_VERTEX_ELEMENTS Format: Opcode

<table>
<tr><th colspan="3">3DSTATE_VERTEX_ELEMENTS</th></tr>
<tr><td></td><td>15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>Vertex Element Count = (DWord Count + 1) / 2<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>1</td><td>DWORD_COUNT_n <b>[Default]</b></td><td>excludes DWords 0,1</td></tr><tr><td>[1,67]</td><td>Range</td><td>1-34 Elements</td></tr></table></td></tr>
<tr><td>1..n</td><td>63:0</td><td><b>Element</b><table><tr><td>Format:</td><td>VERTEX_ELEMENT_STATE</td></tr></table></td></tr>
</table>

3DSTATE_VF_COMPONENT_PACKING

3DSTATE_VF_COMPONENT_PACKING

Source:	RenderCS
Length Bias:	2

This command is used to specify, separately for Vertex Elements [0-31], which post-conversion, 32-bit components are "enabled" to be stored in the URB , and which are "disabled" (not stored). 128 per-component enable bits are provided. Disabling all four components for a given Vertex Element will result in no data stored for that element. Note that any insertion of SGVs (3DSTATE_VF_SGVS) is performed before the packing operation. The **Component Packing Enable** bit (3DSTATE_VF) controls the overall packing process. If that bit is set, the packing process is enabled and the bit mask provided in this command is used to control which components are stored. If that bit is clear, the packing process is disabled - all four components of "valid" Vertex Elements will be stored.

Programming Notes

Programming Restrictions:

- The Vertex Elements referenced in this command correspond to the first 32 VERTEX_ELEMENT structures passed in 3DSTATE_VERTEX_ELEMENTS. A Vertex Element must be marked as "Valid" via 3DSTATE_VERTEX_ELEMENTS or be an SGV or an element between the last valid element and the last SGV in order for the corresponding Component Enable bits of this command to be utilized.
- No enable bits are provided for Vertex Elements [32-33], and therefore no packing is performed on these elements (if Valid, all 4 components are stored).
- If a Vertex Element has Edge Flag Enable set no packing is performed for that element and the corresponding packing state is ignored.
- Component packing is probably only useful for SIMD8 VS thread execution.

At least one component of one "valid" Vertex Element must be enabled.

Software shall enable all components (XYZW) for any and all VERTEX_ELEMENTs associated with a 256-bit SURFACE_FORMAT. It is INVALID to disable any components in these cases.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode

3DSTATE_VF_COMPONENT_PACKING

	23:16	**3D Command Sub Opcode** Default Value: 55h 3DSTATE_VF_COMPONENT_PACKING Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 3h Excludes DWord (0,1) Format: =n
1..4	127:0	**VF Component Packing State Body** Format: 3DSTATE_VF_COMPONENT_PACKING_BODY

3DSTATE_VF_INSTANCING

3DSTATE_VF_INSTANCING

Source:	RenderCS
Length Bias:	2

This command is used to control the "instancing" state associated with a specific vertex element.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 49h 3DSTATE_VF_INSTANCING Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Format: =n Value | Name 1h | Excludes DWord (0,1) **[Default]** 43h | Context Restore
1..2	63:0	**VF Instancing State Body** Format: 3DSTATE_VF_INSTANCING_BODY

3DSTATE_VF

3DSTATE_VF

Source:	RenderCS
Length Bias:	2

This command is used to set various state variables in the VF stage.

The use of the component packing mask is specified via 3DSTATE_VF_COMPONENT_PACKING

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Ch 3DSTATE_VF Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14	**Force Sequential Access Enable** Format: Enable If ENABLED, the VERTEXDATA buffers are accessed sequentially, regardless of the value of 3DPRIMITIVE::VertexAccessType. The VertexID will still be equal to the index obtained from the Index Buffer if 3DPRIMITIVE::VertexAccessType is RANDOM. The3DSTATE_VF_TOPOLOGY::Primitive Topology Type must be set to patchlist 1. If DISABLED, the VERTEXDATA buffers are accessed according to the value of 3DPRIMITIVE::VertexAccessType. **Programming Notes** Software shall not enable both Force Sequential Access Enable AND either of the Sequential/Indexed Draw Cut Index Enable bits. I.e., Force Sequential Access and Cut Index processing are mutually exclusive functions.

<table>
<tr><th colspan="3">3DSTATE_VF</th></tr>
<tr><td></td><td>13</td><td><b>InstanceID Offset Enable</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
If ENABLED, the InstanceID value optionally inserted into the vertex data, and used as an index for vertex element addressing when Instance Stride Enable is ENABLED, is offset by StartInstanceLocation. If DISABLED, InstanceID is not offset by StartInstanceLocation and instead is always 0-based.</td></tr>
<tr><td></td><td>12</td><td><b>Geometry Distribution Enable</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
This bit is used to control whether draw commands are distributed across multiple geometry fixed-function pipelines.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td>Enabled</td><td>Draw commands are distributed across multiple geometry pipelines as controlled by states programmed via 3DSTATE_VFG. When only one geometry pipeline is enabled, or only one geometry pipeline exists, enabling distribution will cause draw commands to be subdivided into instances or batches even though only one geometry pipeline will receive the work.</td></tr>
<tr><td>0</td><td>Disabled</td><td>Draw commands are processed on a draw command basis (i.e., not subdivided into instances or batches), by a single geometry pipeline. Any state programmed via 3DSTATE_VFG is ignored.</td></tr>
</table></td></tr>
<tr><td></td><td>11</td><td><b>VertexID Offset Enable</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
If ENABLED, the VertexID value optionally inserted into the vertex data is offset by StartVertexLocation (SEQUENTIAL draws) or BaseVertexLocation (RANDOM draws).If DISABLED, VertexID is not offset by these values and instead is always 0-based</td></tr>
<tr><td></td><td>10</td><td><b>Sequential Draw Cut Index Enable</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
If ENABLED, vertex indices in SEQUENTIAL 3DPRIMITIVE commands are compared to the Cut Index (specified below). When the vertex index matches the Cut Index any previous topology is terminated.If DISABLED, vertex indices are not compared to the Cut Index. This field can only be enabled for certain primitive topology types. Refer to the table later in this section for details.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Software shall not enable both Force Sequential Access Enable AND either of the Sequential/Indexed Draw Cut Index Enable bits. I.e., Force Sequential Access and Cut Index processing are mutually exclusive functions.</td></tr>
</table></td></tr>
<tr><td></td><td>9</td><td><b>Component Packing Enable</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
If ENABLED, vertex element component packing (as specified by 3DSTATE_VF_COMPONENT_PACKING) is performed before vertices are written into the URB. If DISABLED, no component packing is performed - all components of valid vertex elements will be stored in the URB.</td></tr>
</table>

3DSTATE_VF

	8	**Indexed Draw Cut Index Enable** Format: Enable If ENABLED, vertex indices in RANDOM 3DPRIMITIVE commands are compared to the Cut Index (specified below). When the vertex index matches the Cut Index any previous topology is terminated. If DISABLED, vertex indices are not compared to the Cut Index and are used strictly as indices into vertex buffers. This field can only be enabled for certain primitive topology types. Refer to the table later in this section for details. **Programming Notes** Software shall not enable both Force Sequential Access Enable AND either of the Sequential/Indexed Draw Cut Index Enable bits. I.e., Force Sequential Access and Cut Index processing are mutually exclusive functions.
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n
1	31:0	**VF State Body** Format: 3DSTATE_VF_BODY

3DSTATE_VF_SGVS_2

3DSTATE_VF_SGVS_2

Source:	RenderCS
Length Bias:	2

This command is used to control the insertion of the Extended Parameter (XP0-2) System-Generated Values (SGVs) into an input Vertex URB Entry (VUE) (available as input to a VS thread). The insertions are individually controlled. The insertion locations are specified as 128-bit element locations (starting at the beginning of the VUE) and 32-bit component within those specified elements. The SGV values can be inserted either (a) within a valid vertex element (in which case the value overwrites the value specified via 3DSTATE_VERTEX_ELEMENTS) or (b) beyond the last valid vertex element written to the URB. This permits some orthogonality between the programming of vertex elements (which typically is known at draw time) and programming of SGV insertion (which is associated with the shader). There are some restrictions however (see below). If an SGV is inserted beyond the last valid vertex element, zeroes are first inserted in the VUE after the last valid vertex element up to and including the vertex element receiving an SGV. If both of the SGVs are enabled for insertion, the zeroes will extend to the last vertex element receiving and SGV. Then the SGV(s) are inserted.

The sources for these SGV values are derived from 3DPRIMITIVE command parameters. Controls in the 3DPRIMITIVE command determine whether (a) the parameters are directly defined via in-line command DWords, (b) the parameters are indirectly specified by command stream registers, or (c) the parameters are not included in the 3DPRIMITIVE command and therefore default to 0. Refer to the 3DPRIMITIVE command description for details on these different cases.

The states included in this command are used to (a) enable/disable specific XP* SGV insertions and (b) for XP0 and XP1, specify which 3DPRIMITIVE parameters are used to source the inserted SGV value.

The insertion of SGV values occurs before any component packing (3DSTATE_VF_COMPONENT_PACKING). Therefore the Element Offsets and Component Numbers specified in this command refer to the pre-packed data, following 3DSTATE_VERTEX_ELEMENT processing.

Programming Notes

Programming Restrictions:

- It is INVALID to specify that more than one SGV is to be stored in the same element/component location within the VUE.
- The states programmed by this command overwrite the state programmed by any previous commands. I.e., a specific SGV (if enabled) can only be inserted in one component of a vertex.
- It is INVALID to insert an SGV value past the end of the VUE entry (as determined by VS URB Entry Allocation Size) or past the 33rd vertex element. Therefore the programming of VS URB Entry Allocation Size needs to comprehend any SGV insertion requirements.
- It is INVALID to use this command to overwrite any portion of a 64-bit vertex element component.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode

3DSTATE_VF_SGVS_2		
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 56h 3DSTATE_VF_SGVS_2 Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n
1..2	63:0	**VF SGVS 2 State Body** Format: 3DSTATE_VF_SGVS_2_BODY

3DSTATE_VF_SGVS

3DSTATE_VF_SGVS

Source:	RenderCS
Length Bias:	2

This command is used to control the insertion of the VertexID and InstanceID System-Generated Values (SGVs) into an input Vertex URB Entry (VUE) (available as input to a VS thread). VertexID and InstanceID insertion can be individually controlled. The insertion locations are specified as 128-bit element locations (starting at the beginning of the VUE) and the 32-bit component within those specified elements. The SGV values can be inserted either (a) within a valid vertex element (in which case the value overwrites the value specified via 3DSTATE_VERTEX_ELEMENTS) or (b) beyond the last valid vertex element written to the URB. This permits some orthogonality between the programming of vertex elements (which typically is known at draw time) and programming of SGV insertion (which is associated with the shader). There are some restrictions however (see below). If an SGV is inserted beyond the last valid vertex element, zeroes are first inserted in the VUE after the last valid vertex element up to and including the vertex element receiving an SGV. If both of the SGVs are enabled for insertion, the zeroes will extend to the last (largest index) vertex element receiving an SGV. Then the SGV(s) are inserted.

The insertion of SGV values occurs before any component packing (3DSTATE_VF_COMPONENT_PACKING). Therefore the Element Offsets and Component Numbers specified in this command refer to the pre-packed data, following 3DSTATE_VERTEX_ELEMENT processing.

Programming Notes

Programming Restrictions:

- It is INVALID to store both the VertexID and InstanceID in the same element/component location within the VUE.
- The states programmed by this command overwrite the state programmed by any previous commands. I.e., VertexID and InstanceID (if enabled) can only be inserted in one component of a vertex.
- It is INVALID to insert an SGV value past the end of the VUE entry (as determined by VS URB Entry Allocation Size) or past the 33rd vertex element. Therefore the programming of VS URB Entry Allocation Size needs to comprehend any SGV insertion requirements.
- It is INVALID to use this command to overwrite any portion of a 64-bit vertex element component.
- It is INVALID to use this command to overwrite a EdgeFlag vertex element component or any vertex element beyond it.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode

3DSTATE_VF_SGVS		
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4Ah 3DSTATE_VF_SGVS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n
1	31:0	**VF SGVS State Body** Format: 3DSTATE_VF_SGVS_BODY

3DSTATE_VF_STATISTICS

3DSTATE_VF_STATISTICS

Source:	RenderCS
Length Bias:	1

The VF stage tracks two pipeline statistics, the number of vertices fetched and the number of objects generated. VF will increment the appropriate counter for each when statistics gathering is enabled by issuing the 3DSTATE_VF_STATISTICS command with the [Statistics Enable] bit set.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 1h GFXPIPE_SINGLE_DW Format: Opcode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: Opcode GFXPIPE[28:27 = 1h, 26:24 = 0h, 23:16 = 0Bh] (Pipelined, Single DWord)
	23:16	**3D Command Sub Opcode** Default Value: 0Bh 3DSTATE_VF_STATISTICS Format: Opcode GFXPIPE[28:27 = 1h, 26:24 = 0h, 23:16 = 0Bh] (Pipelined, Single DWord)
	15:1	**Reserved** Access: RO Format: MBZ
	0	**Statistics Enable** Format: Enable If ENABLED, VF will increment the pipeline statistics counters IA_VERTICES_COUNT and IA_PRIMITIVES_COUNT for each vertex fetched and each object output, respectively, for 3DPRIMITIVE commands issued subsequently. If DISABLED, these counters will not be incremented for subsequent 3DPRIMITIVE commands. **Programming Notes** When a 3DPRIMITIVE command with POSH Enable set is executed from the RCS command stream, VF statistics gathering is inhibited for that command.

3DSTATE_VF_TOPOLOGY

3DSTATE_VF_TOPOLOGY

Source:	RenderCS
Length Bias:	2

This command specifies the VF stage's Topology state which can be used to override the Primitive Topology Type in subsequent 3DPRIMITIVE commands.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4Bh 3DSTATE_VF_TOPOLOGY Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n
1	31:0	**VF Topology State Body** Format: 3DSTATE_VF_TOPOLOGY_BODY

3DSTATE_VFG

3DSTATE_VFG

Source:	RenderCS
Length Bias:	2

This command is used to control how objects generated by draw commands are distributed across multiple geometry pipelines (or subdivided if only one geometry pipeline exists or is enabled)via various state variables contained in the VFG stage. The states set by this command are strictly for performance tuning. They will not impact GPU functionality (e.g., no impact to draw command results).

The states set by this command are only used when 3DSTATE_VF::GeometryDistributionEnable is ENABLED (set to 1), which enables distribution across all geometry pipelines. If that state is DISABLED (cleared to 0), only one geometry pipeline will be used.

The highest-level control isDistributionGranularity, where the choices are DrawLevel, InstanceLevel and BatchLevel. DrawLevel causes distribution ofentire draw commands to the geometry pipelines. InstanceLevel causes complete instances with draw commands to be distributed (degenerates to DrawLevel for single-instance draw commands). BatchLevel utilizes a set of "BatchSize" states that are used to specify the size of batches (in vertices) to be distributed as a function of PrimitiveTopology class. Here each instance of draw commands will be divided into one or more batches. BatchLevel will typically provide the best performance.

In addition, a GranularityThresholdEnable state is provided for InstanceLevel and BatchLevel modes. When GranularityThreshold is enabled andInstanceLevel granularity is enabled, DrawLevel distribution may be used for distribution if the total number of vertices in the draw command is very small. Likewise, when GranularityThreshold is enabled and BatchLevel granularity is enabled, DrawLevel distribution may be used if the total number of vertices in the draw command is very small, or InstanceLevel distribution may be used if the total number of vertices in each instance is very small.

Programming Notes
The maximum batch size must less than 4K vertices.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 57h 3DSTATE_VFG Format: OpCode

3DSTATE_VFG

	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n
1..3	95:0	**VFG State Body** Format: 3DSTATE_VFG_BODY

3DSTATE_VIEWPORT_STATE_POINTERS_CC

3DSTATE_VIEWPORT_STATE_POINTERS_CC

Source:	RenderCS
Length Bias:	2

The 3DSTATE_VIEWPORT_STATE_POINTERS_CC command is used to define the location of fixed functions' viewport state table.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 23h 3DSTATE_VIEWPORT_STATE_POINTERS_CC Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Viewport State Pointers CC State Body** Format: 3DSTATE_VIEWPORT_STATE_POINTERS_CC_BODY

3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP

3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP

Source:	RenderCS
Length Bias:	2

The 3DSTATE_VIEWPORT_STATE_POINTERS_CLIP command is used to define the location of fixed functions' viewport state table.

Restriction
When executed in POCS command stream, this programs viewport state of the POCS pipeline.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 21h 3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:0	**Viewport State Pointers SF Clip State Body** Format: 3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP_BODY

3DSTATE_VS

3DSTATE_VS		
Source:	RenderCS, PositionCS	
Length Bias:	2	
This command specifies most of the state used by the Vertex Shader (VS) stage.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 10h 3DSTATE_VS Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 7h Excludes DWord (0,1) Format: =n
1..8	255:0	**VS State Body** Format: 3DSTATE_VS_BODY

3DSTATE_WM_CHROMAKEY

3DSTATE_WM_CHROMAKEY

Source:	RenderCS
Length Bias:	2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4Ch 3DSTATE_WM__CHROMAKEY Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**Dword Length** Default Value: 0h Excludes Dword (0,1) Format: =n Total Length - 2
1	31:0	**WM Chromakey State Body** Format: **3DSTATE_WM_CHROMAKEY_BODY**

3DSTATE_WM_DEPTH_STENCIL

3DSTATE_WM_DEPTH_STENCIL

Source:	RenderCS
Length Bias:	2

This command replaces the indirect state DEPTH_STENCIL_STATE with an inline state command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4Eh 3DSTATE_WM_DEPTH_STENCIL Format: OpCode
	15:13	**Reserved** Access: RO Format: MBZ
	12	**Depth State Modify Disable** Format: Disable When this bit is set, the following fields will be ignored: • Depth Test Function • Depth Test Enable • Depth Buffer Write Enable
	11	**Stencil State Modify Disable** Format: Disable When this bit is set, the following fields will be ignored: • Stencil Fail Op • Stencil Pass Depth Fail Op • Stencil Pass Depth Pass Op • Backface Stencil Test Function • Backface Stencil Fail Op • Backface Stencil Pass Depth Fail Op

3DSTATE_WM_DEPTH_STENCIL		
		• Backface Stencil Pass Depth Pass Op • Stencil Test Function • Double Sided Stencil Enable • Stencil Test Enable • Stencil Buffer Write Enable
	10	**Stencil Write Mask Modify Disable** Format: | Disable When this bit is set, the following fields will be ignored: • Stencil Write Mask • Backface Stencil Write Mask
	9	**Stencil Test Mask Modify Disable** Format: | Disable When this bit is set, the following fields will be ignored: • Stencil Test Mask • Backface Stencil Test Mask
	8	**Stencil Reference Value Modify Disable** Format: | Disable When this bit is set, the following fields will be ignored: • Stencil Reference Value • Backface Stencil Reference Value
	7:0	**Dword Length** Default Value: | 02h Excludes Dword (0,1) Format: | =n Total Length - 2
1..3	95:0	**WM Depth Stencil State Body** Format: | 3DSTATE_WM_DEPTH_STENCIL_BODY

3DSTATE_WM_HZ_OP

3DSTATE_WM_HZ_OP

Source:	RenderCS
Length Bias:	2

This command provides for clearing Z and/or stencil or resolving either HZ buffer or Z buffer.

Programming Notes
As this command generates an implicit rectangle, SW must make sure any MMIO register writes following WM_HZ_OP must be preceded by **PIPE_CONTROL** with **Command Streamer Stall Enable** bit set.
3DSTATE_DRAWING_RECTANGLE must be programmed such that it does not clip the HZ_OP command's rectangle. See programming notes for X/Y Min and X/Y Max below. 3DSTATE_DRAWING_RECTANGLE command must come before 3DSTATE_WM_HZ_OP in the command buffer **Caution:** There is a difference in how X/Y coordinates are interpreted by 3DSTATE_DRAWING_RECTANGLE vs. 3DSTATE_WM_HZ_OP. HZ_OP rectangle parameters are exclusive on max side, for example to have 8x4 rectangle we would program X Min = 0, Y Min = 0, X Max = 8, Y Max = 4. Draw Rectangle parameters are inclusive on max side, meaning, for 8x4 rectangle the values would be X Min = 0, Y Min = 0, X Max = 7, Y Max = 3.
3DSTATE_MULTISAMPLE packet must be used prior to this packet to change the Number of Multisamples. This packet must not be used to change Number of Multisamples in a rendering sequence. 3DSTATE_RASTER if used must be programmed prior to using this packet.
Since HZ_OP has to be sent twice (first time set the clear/resolve state and 2nd time to clear the state), and HW internally flushes the depth cache on HZ_OP, there is no need to explicitly send a Depth Cache flush after Clear or Resolve.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 52h 3DSTATE_WM_HZ_OP Format: OpCode
	15:9	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">3DSTATE_WM_HZ_OP</th></tr>
<tr><td></td><td>8</td><td><b>Predicate Enable</b>
If set, this command is executed (or not) depending on the current value of the MI Predicate internal state bit (MI_PREDICATE_RESULT). This command is ignored only if PredicateEnable is set and the Predicate state (MI_PREDICATE_RESULT[0])) bit is 0.
This command is un-conditionally NOOP'd when MI_SET_PREDICATE_RESULT[0] is set.</td></tr>
<tr><td></td><td>7:0</td><td><b>Dword Length</b>
<table><tr><td>Format:</td><td>=n</td></tr></table>
Total Length - 2
<table><tr><th>Value</th><th>Name</th></tr><tr><td>03h</td><td>Excludes Dword (0,1) <b>[Default]</b></td></tr></table></td></tr>
<tr><td>1..5</td><td>159:0</td><td><b>WM HZ OP State Body</b>
<table><tr><td>Format:</td><td>3DSTATE_WM_HZ_OP_BODY</td></tr></table></td></tr>
</table>

3DSTATE_WM

3DSTATE_WM		
Source:	RenderCS	
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 14h 3DSTATE_WM Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:0	**WM State Body** Format: 3DSTATE_WM_BODY

A64 Byte Scattered Read MSD

MSD1R_A64_BS - A64 Byte Scattered Read MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 10h Format: Opcode A64 Scattered Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 0h Format: Opcode Byte Read/Write subtype
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Byte Scattered Write MSD

MSD1W_A64_BS - A64 Byte Scattered Write MSD

Source: EuSubFunctionDataPort1

Length Bias: 1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Ah Format: Opcode A64 Scattered Write message
	13	**Reserved** Access: RO Format: MBZ
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 0h Format: Opcode Byte Read/Write subtype

MSD1W_A64_BS - A64 Byte Scattered Write MSD		
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Scattered Read MSD

MSD1R_A64_DWS - A64 Dword Scattered Read MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 10h Format: Opcode A64 Scattered Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 1h Format: Opcode Dword Read/Write subtype
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Scattered Write MSD

MSD1W_A64_DWS - A64 Dword Scattered Write MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Ah Format: Opcode A64 Scattered Write message
	13	**Reserved** Access: RO Format: MBZ
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 1h Format: Opcode Dword Read/Write subtype

MSD1W_A64_DWS - A64 Dword Scattered Write MSD		
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Untyped Atomic Float with Return Data Operation MSD

MSD1R_A64_DWAF - A64 Dword Untyped Atomic Float with Return Data Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Dh Format: Opcode A64 Untyped Atomic Float Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2S Only SIMD8 operations are supported.
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10:8	**Atomic Float Operation** Format: MDC_FOP Specifies the atomic float operation to be performed.

	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Dword Untyped Atomic Float Write Only Operation MSD

MSD1W_A64_DWAF - A64 Dword Untyped Atomic Float Write Only Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: **MDC_MHF** Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Dh Format: Opcode A64 Untyped Atomic Float Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: **MDC_SM2S** **Description** Only SIMD8 operations are supported.
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.

MSD1W_A64_DWAF - A64 Dword Untyped Atomic Float Write Only Operation MSD

	10:8	**Atomic Float Operation Type** Format: **MDC_FOP** Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: **MDC_STATELESS** Specifies the message is stateless

A64 Dword Untyped Atomic Integer with Return Data Operation MSD

MSD1R_A64_DWAI - A64 Dword Untyped Atomic Integer with Return Data Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP Specifies the atomic integer operation to be performed.

MSD1R_A64_DWAI - A64 Dword Untyped Atomic Integer with Return Data Operation MSD

	7:0	**Binding Table Index** Format: **MDC_STATELESS** Specifies the message is stateless

A64 Dword Untyped Atomic Integer Write Only Operation MSD

MSD1W_A64_DWAI - A64 Dword Untyped Atomic Integer Write Only Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Hword Block Read MSD

MSD1R_A64_HWB - A64 Hword Block Read MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 14h Format: Opcode A64 Oword Block Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:11	**A64 Block Message Subtype** Default Value: 3h Format: Opcode Hword Block Read/Write subtype
	10:8	**Data Elements** Format: MDC_A64_DB_HW Specifies the number of contiguous Hwords to be read or written
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Hword Block Write MSD

MSD1W_A64_HWB - A64 Hword Block Write MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 15h Format: Opcode A64 Hword Block Write message
	13	**Reserved** Access: RO Format: MBZ
	12:11	**A64 Block Message Subtype** Default Value: 3h Format: Opcode Hword Block Read/Write subtype
	10:8	**Data Elements** Format: MDC_A64_DB_HW Specifies the number of contiguous Hwords to be read or written
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Oword Aligned Block Read MSD

MSD1R_A64_OWAB - A64 Oword Aligned Block Read MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 14h Format: Opcode A64 Oword Block Read message
	13	**Reserved** Access: RO Format: MBZ
	12:11	**A64 Block Message Subtype** Default Value: 1h Format: Opcode Oword Aligned Block Read subtype
	10:8	**Data Elements** Format: MDC_A64_DB_OW Specifies the number of contiguous Owords to be read
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Oword Aligned Block Write MSD

MSD1W_A64_OWAB - A64 Oword Aligned Block Write MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 15h Format: Opcode A64 Oword Block Write message
	13	**Reserved** Access: RO Format: MBZ
	12:11	**A64 Block Message Subtype** Default Value: 1h Format: Opcode Oword Aligned Block Write subtype
	10:8	**Data Elements** Format: MDC_A64_DB_OW Specifies the number of contiguous Owords to be written
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Oword Block Read MSD

MSD1R_A64_OWB - A64 Oword Block Read MSD

Source: EuSubFunctionDataPort1

Length Bias: 1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 14h Format: Opcode A64 Oword Block Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:11	**A64 Block Message Subtype** Default Value: 0h Format: Opcode Oword Block Read/Write subtype
	10:8	**Data Elements** Format: MDC_A64_DB_OW Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Oword Block Write MSD

MSD1W_A64_OWB - A64 Oword Block Write MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 15h Format: Opcode A64 Oword Block Write message
	13	**Reserved** Access: RO Format: MBZ
	12:11	**A64 Block Message Subtype** Default Value: 0h Format: Opcode Oword Block Read/Write subtype
	10:8	**Data Elements** Format: MDC_A64_DB_OW Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Page CCS Update Operation MSD

MSD_A64_CCS_PG_OP - A64 Page CCS Update Operation MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Default Value: 0 Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 17h Format: Opcode A64 Page CCS Update Operation Message.
	13:10	**Reserved** Access: RO Format: MBZ
	9:8	**CCS Page Update Opcode** Format: MDC_CCS_PG_OP Specifies the opcode for CCS Update operation.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword Scattered Read MSD

MSD1R_A64_QWS - A64 Qword Scattered Read MSD

Source: EuSubFunctionDataPort1

Length Bias: 1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 10h Format: Opcode A64 Scattered Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 2h Format: Opcode Qword Read/Write subtype
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Qword Scattered Write MSD

MSD1W_A64_QWS - A64 Qword Scattered Write MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Ah Format: Opcode A64 Scattered Write message
	13	**Reserved** Access: RO Format: MBZ
	12	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	11:10	**Data Elements** Format: MDC_A64_DS Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 2h Format: Opcode Qword Read/Write subtype

MSD1W_A64_QWS - A64 Qword Scattered Write MSD		
	7:0	**Binding Table Index** Format: **MDC_STATELESS** Specifies the message is stateless

A64 Qword Untyped Atomic Integer with Return Data Operation MSD

MSD1R_A64_QWAI - A64 Qword Untyped Atomic Integer with Return Data Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP Specifies the atomic integer operation to be performed.

MSD1R_A64_QWAI - A64 Qword Untyped Atomic Integer with Return Data Operation MSD

	7:0	**Binding Table Index** Format: | MDC_STATELESS Specifies the message is stateless

A64 Qword Untyped Atomic Integer Write Only Operation MSD

MSD1W_A64_QWAI - A64 Qword Untyped Atomic Integer Write Only Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 12h Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Untyped Surface CCS Operation MSD

MSD_A64_US_CCS_OP - A64 Untyped Surface CCS Operation MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 18h Format: Opcode A64 Untyped Surface CCS update operation.
	13:12	**SIMD Mode** Format: MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**CCS Operation** Format: MDC_CCS_SEC_OP Specifies which CCS operation is performed.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Untyped Surface Read MSD

MSD1R_A64_US - A64 Untyped Surface Read MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 11h Format: Opcode A64 Untyped Surface Read message
	13:12	**SIMD Mode** Format: MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: MDC_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Untyped Surface Uncompressed Write MSD

MSD_A64_US_UCW - A64 Untyped Surface Uncompressed Write MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 19h Format: Opcode A64 Untyped Surface Uncompressed Write message
	13:12	**SIMD Mode** Format: MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: MDC_UW_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Untyped Surface Write MSD

MSD1W_A64_US - A64 Untyped Surface Write MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 19h Format: Opcode A64 Untyped Surface Write message
	13:12	**SIMD Mode** Format: MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: MDC_UW_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: MDC_STATELESS Specifies the message is stateless

A64 Word Untyped Atomic Float with Return Data Operation MSD

MSD1R_A64_WAF - A64 Word Untyped Atomic Float with Return Data Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Eh Format: Opcode A64 Untyped Atomic Half Float Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Access: RO Format: MBZ
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10:8	**Atomic Float Operation** Format: MDC_FOP Specifies the atomic float operation to be performed.

MSD1R_A64_WAF - A64 Word Untyped Atomic Float with Return Data Operation MSD		
	7:0	**Binding Table Index** Format: **MDC_STATELESS** Specifies the message is stateless

A64 Word Untyped Atomic Float Write Only Operation MSD

MSD1W_A64_WAF - A64 Word Untyped Atomic Float Write Only Operation MSD

Source: EuSubFunctionDataPort1

Length Bias: 1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Eh Format: Opcode A64 Untyped Atomic Half Float Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Access: RO Format: MBZ
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10:8	**Atomic Float Operation Type** Format: MDC_FOP Specifies the atomic float operation to be performed.

MSD1W_A64_WAF - A64 Word Untyped Atomic Float Write Only Operation MSD

	7:0	**Binding Table Index** Format: | MDC_STATEESS Specifies the message is stateless

A64 Word Untyped Atomic Integer with Return Data Operation MSD

MSD1R_A64_WAI - A64 Word Untyped Atomic Integer with Return Data Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Half Integer Operation message
	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 16-bit integers
	11:8	**Atomic Integer Operation** Format: MDC_AOP Specifies the atomic integer operation to be performed.

MSD1R_A64_WAI - A64 Word Untyped Atomic Integer with Return Data Operation MSD

	7:0	**Binding Table Index** Format: | MDC_STATELESS Specifies the message is stateless

A64 Word Untyped Atomic Integer Write Only Operation MSD

MSD1W_A64_WAI - A64 Word Untyped Atomic Integer Write Only Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: **MDC_MHF** The message forbids a header
	18:14	**Message Type** Default Value: 13h Format: Opcode A64 Untyped Atomic Half Integer Operation message
	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Format: Opcode Operations are on 16-bit integers
	11:8	**Atomic Integer Operation** Format: **MDC_AOP** Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: **MDC_STATELESS** Specifies the message is stateless

Addition

add - Addition

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

The add instruction performs component-wise addition of src0 and src1 and stores the results in dst. Addition of two floating-point numbers follows rules in add (IEEE mode) or add (ALT mode).

Floating-Point Addition of A (Column) and B (Row) in IEEE Mode

	-inf	**-finite**	**-denorm**	**-0**	**+0**	**+denorm**	**+finite**	**+inf**	**NaN**
-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf	NaN	NaN
-finite	-inf	*	A	A	A	A	**	+inf	NaN
-denorm	-inf	B	-0/-denorm/-finite^	-0/-denorm^	+0/+denorm^	+0/+denorm/-denorm^	B	+inf	NaN
-0	-inf	B	-0/denorm^	-0	+0	+0/+denorm	B	+inf	NaN
+0	-inf	B	+0/+denorm^	+0	+0	+0/+denorm	B	+inf	NaN
+denorm	-inf	B	+0/+denorm/-denorm^	+0/+denorm^	+0/+denorm^	+0/+denorm/+finite^	B	+inf	NaN
+finite	-inf	**	A	A	A	A	***	+inf	NaN
+inf	NaN	+inf	+inf	+inf	+inf	+inf	+inf	+inf	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Notes:									
^	Non-zero results are applicable when denorm is enabled.								
*	Result can be {–finite}.								
**	Result can be {–finite, –0, +0, +finite}.								
***	Result can be {+finite}.								

Floating-Point Addition of A (Column) and B (Row) in ALT Mode

	-fmax	**-finite**	**-denorm**	**-0**	**+0**	**+denorm**	**+finite**	**+fmax**	********
-fmax	-fmax	-fmax	-fmax	-fmax	-fmax	-fmax	-finite	+0	
-finite	-fmax	*	A	A	A	A	**	+fmax	
-denorm	-fmax	B	-0	-0	+0	+0	B	+fmax	
-0	-fmax	B	-0	-0	+0	+0	B	+fmax	
+0	-fmax	B	+0	+0	+0	+0	B	+fmax	
+denorm	-fmax	B	+0	+0	+0	+0	B	+fmax	
+finite	-finite	**	A	A	A	A	***	+fmax	
+fmax	+0	+fmax	+fmax	+fmax	+fmax	+fmax	+fmax	+fmax	

add - Addition

****										
Notes:										
*	Result can be {–fmax, -finite}.									
**	Result can be {–finite, –0, +0, +finite}.									
***	Result can be {+fmax, +finite}.									
****	Result is undefined if A or B is {-inf, +inf, NaN}.									

Format:

```
[(pred)] add[.cmod] (exec_size) dst src0 src1
```

Programming Notes

Use a source modifier with add to implement subtraction.

Restriction

Pure bfloat operation is not supported.

Syntax

[(pred)] add[.cmod] (exec_size) reg reg reg
[(pred)] add[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn);

 for ( n = 0; n < exec_size; n++ ) {
   if ( WrEn.chan[n] ) {
     dst.chan[n] = src0.chan[n] + src1.chan[n];
   }
 }
```

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
F	F
HF	HF
BF, F	BF, F

DWord	Bit	Description	
0..3	127:126	**Reserved**	
		Exists If:	([Src1.IsImm]==false)
		Format:	MBZ
	127:96	**Src1.ImmValue[31:0]**	
		Exists If:	([Src1.IsImm]==true)

add - Addition

125:122	**Reserved**		
		Exists If:	([Src1.IsImm]==false)
		Format:	MBZ
121:120	**Src1.Mod**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**SrcMod**
119:116	**Src1.VertStride**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**VertStride**
115:113	**Src1.Width**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**Width**
112	**Src1.AddrMode**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**AddrMode**
111:98	**Src1.Operand**		
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
		Format:	**IndirectOperand**
111:98	**Src1.Operand**		
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
		Format:	**DirectOperand**
97:96	**Src1.HorzStride**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**HorzStride**
95:92	**CondCtrl**		
		Format:	**FlagModifier**
91:88	**Src1.DataType**		
		Exists If:	([Src1.IsImm]==true)
		Format:	**ImmDataType**
91:88	**Src1.DataType**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**RegDataType**
87:84	**Src0.VertStride**		
		Format:	**VertStride**
83:81	**Src0.Width**		
		Format:	**Width**

<table>
<tr><th colspan="3">add - Addition</th></tr>
<tr><td></td><td>80</td><td><b>Src0.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src1.IsImm</b>
This field indicate that Source 1 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">add - Addition</th></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>No Compaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields</td></tr>
</table>

<table>
<tr><th colspan="3">add - Addition</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td><b>Header</b></td></tr></table></td></tr>
</table>

Addition Ternary

add3 - Addition Ternary

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

The add3 instruction takes adds three sources and then stores the final results in dst.

Format:

```
        [(pred)] add3 (exec_size) [cmod] dst src0 src1 src2
```

Restriction

All three-source instructions have certain restrictions, described in *Instruction Formats*.

Syntax

```
[(pred)] add3 (exec_size) [cmod] reg reg reg reg
 [(pred)] add3 (exec_size) [cmod] reg reg reg imm16
 [(pred)] add3 (exec_size) [cmod] reg imm16 reg reg
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = src0.chan[n] + src1.chan[n] + src2.chan[n];
     }
 }
```

Src Types	Dst Types
*W,*D	*W,*D

DWord	Bit	Description
0..3	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) AND ([Header][Opcode]!=madm) Format: **DirectOperand**
	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) AND ([Header][Opcode]==madm) Format: **MacroOperand**
	127:112	**Src2.ImmValue[15:0]** Exists If: ([Src2.IsImm]==true)

add3 - Addition Ternary

	113:112	**Src2.HorzStride**	Exists If:	([Src2.IsImm]==false)
			Format:	**HorzStride**
	111:98	**Src1.Operand**	Exists If:	([Header][Opcode]!=madm)
			Format:	**DirectOperand**
	111:98	**Src1.Operand**	Exists If:	([Header][Opcode]==madm)
			Format:	**MacroOperand**
	97:96	**Src1.HorzStride**	Format:	**HorzStride**
	95:92	**CondCtrl**	Format:	**FlagModifier**
	91	**Src1.VertStride[1]**	Format:	**TernaryVertStride[1:1]**
	90:88	**Src1.DataType**	Format:	**TernaryDataType**
	87:86	**Src1.Mod**	Format:	**SrcMod**
	85:84	**Src2.Mod**	Format:	**SrcMod**
	83	**Src1.VertStride[0]**	Format:	**TernaryVertStride[0:0]**
	82:80	**Src2.DataType**	Format:	**TernaryDataType**
	79:66	**Src0.Operand**	Exists If:	([Src0.IsImm]==false) AND ([Header][Opcode]!=madm)
			Format:	**DirectOperand**
	79:66	**Src0.Operand**	Exists If:	([Src0.IsImm]==false) AND ([Header][Opcode]==madm)
			Format:	**MacroOperand**
	79:64	**Src0.ImmValue[15:0]**	Exists If:	([Src0.IsImm]==true)
	65:64	**Src0.HorzStride**	Exists If:	([Src0.IsImm]==false)
			Format:	**HorzStride**

<table>
<tr><th colspan="3">add3 - Addition Ternary</th></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Header][Opcode]!=madm)</td></tr><tr><td>Format:</td><td>DirectOperand</td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Header][Opcode]==madm)</td></tr><tr><td>Format:</td><td>MacroOperand</td></tr></table></td></tr>
<tr><td></td><td>49</td><td><b>Reserved</b><table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>48</td><td><b>Dst.HorzStride</b>
This field provides the distance in unit of data elements between two adjacent data elements within a row (horizontal) in the register region for the operand.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>1 element</td></tr><tr><td>1</td><td>2 element</td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src2.IsImm</b>
This field indicate that Source 2 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td>SrcMod</td></tr></table></td></tr>
<tr><td></td><td>43</td><td><b>Src0.VertStride[1]</b><table><tr><td>Format:</td><td>TernaryVertStride[1:1]</td></tr></table></td></tr>
<tr><td></td><td>42:40</td><td><b>Src0.DataType</b><table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
<tr><td></td><td>39</td><td><b>ExecDataType</b>
This field indicate the datatype mode of ternary instruction. Integer or Float.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Integer</td></tr><tr><td>1</td><td>Float</td></tr></table></td></tr>
<tr><td></td><td>38:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
</table>

add3 - Addition Ternary

35 **Src0.VertStride[0]**

Format:	**TernaryVertStride[0:0]**

34 **Saturate**

Format:	**Saturate**

33 **AccWrCtrl**

Format:	**AccWrCtrl**

32 **AtomicCtrl**

Format:	**AtomicCtrl**

31 **MaskCtrl**

Mask Control (formerly Write Enable Control). This field determines if the the per channel write enables are used to generate the final write enable. This field should be normally "0".

Value	Name	Description
0	Normal **[Default]**	Normal. Per channel write enable used for final write enable generation.
1	NoMask	NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.

30 **Reserved**

29 **CmptCtrl**

Format:	MBZ

Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.

Value	Name	Description
0	NoCompaction **[Default]**	No compaction. 128-bit native instruction supporting all instruction options.
1	Compacted	Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.

28 **PredInv**

This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields

Value	Name	Description
0	Positive **[Default]**	Positive polarity of predication. Use the predication mask produced by PredCtrl.

add3 - Addition Ternary

		1 Negative Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.
	27:24	**PredCtrl** Format: PredCtrl This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.
	23	**FlagRegNum[0]** This field specifies bit[0] of the register number for a flag register operand.
	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: ChanOff This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: ExecSize This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: Header

Addition with Carry

addc - Addition with Carry

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	false
Source Modifier:	false

The addc instruction performs component-wise addition of src0 and src1 and stores the results in dst; it also stores the carry into acc. If the operation produces a carry out, 0x00000001 is stored in acc, else 0x00000000 is stored in acc.

Format:

```
[(pred)] addc[.cmod] (exec_size) dst src0 src1
```

Restriction
AccWrEn is required.
The accumulator is an implicit destination and thus cannot be an explicit destination operand.

Syntax

```
[(pred)] addc[.cmod] (exec_size) reg reg reg
 [(pred)] addc[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = src0.chan[n] + src1.chan[n];
         acc.chan[n] = carry(src0.chan[n] + src1.chan[n]);
     }
 }
```

Src Types	Dst Types
UD	UD

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)

addc - Addition with Carry

	125:122	**Reserved**	
		Exists If:	([Src1.IsImm]==false)
		Format:	MBZ
	121:120	**Src1.Mod**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**SrcMod**
	119:116	**Src1.VertStride**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**VertStride**
	115:113	**Src1.Width**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**Width**
	112	**Src1.AddrMode**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**AddrMode**
	111:98	**Src1.Operand**	
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
		Format:	**IndirectOperand**
	111:98	**Src1.Operand**	
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
		Format:	**DirectOperand**
	97:96	**Src1.HorzStride**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**HorzStride**
	95:92	**CondCtrl**	
		Format:	**FlagModifier**
	91:88	**Src1.DataType**	
		Exists If:	([Src1.IsImm]==true)
		Format:	**ImmDataType**
	91:88	**Src1.DataType**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**RegDataType**
	87:84	**Src0.VertStride**	
		Format:	**VertStride**
	83:81	**Src0.Width**	
		Format:	**Width**

addc - Addition with Carry

	80	**Src0.AddrMode**	
		Format:	**AddrMode**
	79:66	**Src0.Operand**	
		Exists If:	([Src0.AddrMode]==Direct)
		Format:	**DirectOperand**
	79:66	**Src0.Operand**	
		Exists If:	([Src0.AddrMode]==Indirect)
		Format:	**IndirectOperand**
	65:64	**Src0.HorzStride**	
		Format:	**HorzStride**
	63:50	**Dst.Operand**	
		Exists If:	([Dst.AddrMode]==Direct)
		Format:	**DirectOperand**
	63:50	**Dst.Operand**	
		Exists If:	([Dst.AddrMode]==Indirect)
		Format:	**IndirectOperand**
	49:48	**Dst.HorzStride**	
		Format:	**HorzStride**
	47	**Src1.IsImm** This field indicate that Source 1 operand is carrying an immediate value.	
		Value	**Name**
		0	false **[Default]**
		1	true
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value.	
		Value	**Name**
		0	false **[Default]**
		1	true
	45:44	**Src0.Mod**	
		Format:	**SrcMod**
	43:40	**Src0.DataType**	
		Exists If:	([Src0.IsImm]==false)
		Format:	**RegDataType**
	43:40	**Src0.DataType**	
		Exists If:	([Src0.IsImm]==true)
		Format:	**ImmDataType**

addc - Addition with Carry

<table>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields</td></tr>
</table>

addc - Addition with Carry

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table>
</td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>
This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b>
<table><tr><td>Format:</td><td><b>Header</b></td></tr></table>
</td></tr>
</table>

Arithmetic Shift Right

asr - Arithmetic Shift Right

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

Perform component-wise arithmetic right shift of the bits in src0 by the shift count indicated in src1, storing the results in dst. If src0 has a signed type, insert copies of src0's sign bit in the number of MSBs indicated by the shift count. Otherwise insert 0 bits. When src0 is accumulator and/or source modifier is used with src0 the sign bit is inserted in MSBs which come from the additional precision. **Note**: For Word and DWord operands, the accumulators have 33 bits.

In QWord mode, the shift count is taken from the low six bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 63. Otherwise the shift count is taken from the low five bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 31. The operation uses QWord mode if src0 or dst has the Q or UQ type but not if src1 is the only operand with the Q or UQ type. For positive values, this operation is src0 / 2shiftCount and for negative values, this operation is src0 / 2shiftCount - 1.

Format:

```
[(pred)] asr[.cmod] (exec_size) dst src0 src1
```

Programming Notes

Programming Notes
If src0 is -1, the result is -1 regardless of the shift count.
For unsigned src0 types, asr and shr produce the same result.

Syntax

```
[(pred)] asr[.cmod] (exec_size) reg reg reg
 [(pred)] asr[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.channel[n] ) {
         shiftCnt = src0 or dst has Q or UQ type ? src1.chan[n] & 0x3F : src1.chan[n] &
0x1F
         if (src0.chan[n] >= 0) {
             dst.chan[n] = src0.chan[n] » shiftCnt;
         } else {
             int maskLSB = pow(2, shiftCnt) - 1;
             if ( maskLSB & src0.chan[n] == 0 ) {
                 dst.chan[n] = sign(src0.chan[n]) * ((abs)src0.chan[n] » shiftCnt);
             } else {
                 dst.chan[n] = sign(src0.chan[n]) * ((abs)src0.chan[n] » shiftCnt) - 1;
             }
         }
     }
```

asr - Arithmetic Shift Right

```
}
```

Src Types	Dst Types
*B,*W,*D	*B,*W,*D

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)
	125:122	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	121:120	**Src1.Mod** Exists If: ([Src1.IsImm]==false) Format: **SrcMod**
	119:116	**Src1.VertStride** Exists If: ([Src1.IsImm]==false) Format: **VertStride**
	115:113	**Src1.Width** Exists If: ([Src1.IsImm]==false) Format: **Width**
	112	**Src1.AddrMode** Exists If: ([Src1.IsImm]==false) Format: **AddrMode**
	111:98	**Src1.Operand** Exists If: ([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect) Format: **IndirectOperand**
	111:98	**Src1.Operand** Exists If: ([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct) Format: **DirectOperand**
	97:96	**Src1.HorzStride** Exists If: ([Src1.IsImm]==false) Format: **HorzStride**
	95:92	**CondCtrl** Format: **FlagModifier**

asr - Arithmetic Shift Right

	91:88	**Src1.DataType**	
		Exists If:	([Src1.IsImm]==true)
		Format:	**ImmDataType**
	91:88	**Src1.DataType**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**RegDataType**
	87:84	**Src0.VertStride**	
		Format:	**VertStride**
	83:81	**Src0.Width**	
		Format:	**Width**
	80	**Src0.AddrMode**	
		Format:	**AddrMode**
	79:66	**Src0.Operand**	
		Exists If:	([Src0.AddrMode]==Direct)
		Format:	**DirectOperand**
	79:66	**Src0.Operand**	
		Exists If:	([Src0.AddrMode]==Indirect)
		Format:	**IndirectOperand**
	65:64	**Src0.HorzStride**	
		Format:	**HorzStride**
	63:50	**Dst.Operand**	
		Exists If:	([Dst.AddrMode]==Direct)
		Format:	**DirectOperand**
	63:50	**Dst.Operand**	
		Exists If:	([Dst.AddrMode]==Indirect)
		Format:	**IndirectOperand**
	49:48	**Dst.HorzStride**	
		Format:	**HorzStride**
	47	**Src1.IsImm**	
		This field indicate that Source 1 operand is carrying an immediate value.	
		Value	**Name**
		0	false **[Default]**
		1	true
	46	**Src0.IsImm**	
		This field indicate that Source 0 operand is carrying an immediate value.	

<table>
<tr><th colspan="3">asr - Arithmetic Shift Right</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td><b>Saturate</b></td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td><b>AccWrCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.</td></tr>
</table>

asr - Arithmetic Shift Right

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table>
</td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table>
</td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td><b>PredCtrl</b></td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table>
<tr><td>Format:</td><td><b>ChanOff</b></td></tr>
</table>
This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table>
<tr><td>Format:</td><td><b>ExecSize</b></td></tr>
</table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b>
<table>
<tr><td>Format:</td><td><b>Header</b></td></tr>
</table>
</td></tr>
</table>

Atomic Add

DP_ATOMIC_ADD - Atomic Add

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Atomic signed int add of src1 from memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.ADD.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base for
(n = 0; n < 32; n++) { if (Msg.ChEn[n]) { old =
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = old +
src1[0].data_size[n]; dest[0].data_size[n] = old; } }
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_ATOMIC_ADD - Atomic Add

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

<table>
<tr><th colspan="3">DP_ATOMIC_ADD - Atomic Add</th></tr>
<tr><td></td><td>11:9</td><td><b>Data Size</b>
<table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>
Specifies both bit size of the data payload item in memory and the bit size used in the register payload.
<table>
<tr><th>Restriction</th><th>Source</th></tr>
<tr><td>Restriction : D64 atomic operations are not supported on SFID_E (SLM) and SFID_D (TGM).</td><td>SFID_D, SFID_E</td></tr>
<tr><td>Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLOAT.</td><td>SFID_D</td></tr>
<tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr>
</table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b>
<table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>
Specifies the bit size of each address payload item.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Atomic Operation</b>
<table><tr><td>Default Value:</td><td>12 Atomic Add</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Atomic AND

DP_ATOMIC_AND - Atomic AND

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Atomic store the bitwise AND of src1 and memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.AND.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
for (n = 0; n < 32; n++) {
    if (Msg.ChEn[n]) {
        old = ((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
        ((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = old & src1[0].data_size[n];
        dest[0].data_size[n] = old;
    }
}
```

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31</td><td><b>Reserved</b>
Access: RO
Format: MBZ</td></tr>
<tr><td></td><td>30:29</td><td><b>Address Type</b>
Format: DP_ADDR_SURFACE_TYPE
Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc.
<b>Restriction</b>
Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.</td></tr>
<tr><td></td><td>28:25</td><td><b>Src0 Length</b>
Format: DP_ADDR_REG_SIZE
Specifies the size of the address payload, in registers.</td></tr>
</table>

DP_ATOMIC_AND - Atomic AND

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16.
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16.

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

<table>
<tr><th colspan="3">DP_ATOMIC_AND - Atomic AND</th></tr>
<tr><td></td><td>11:9</td><td><b>Data Size</b>
<table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>
Specifies both bit size of the data payload item in memory and the bit size used in the register payload.
<table>
<tr><th>Restriction</th><th>Source</th></tr>
<tr><td>Restriction : D64 atomic operations are not supported on SFID_E (SLM) and SFID_D (TGM).</td><td>SFID_D, SFID_E</td></tr>
<tr><td>Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLOAT.</td><td>SFID_D</td></tr>
<tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr>
</table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b>
<table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>
Specifies the bit size of each address payload item.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Atomic Operation</b>
<table><tr><td>Default Value:</td><td>24 Atomic AND</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Atomic Compare Exchange

DP_ATOMIC_CMPXCHG - Atomic Compare Exchange

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Atomic bit-compare src1_X and memory data and replace if equal with src1_Y. Returns the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size. The src1 data payload is a vector of 2 values (X, Y). X is the value to match, and Y is the value to replace it with.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.CMPXCHG.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base for
(n = 0; n < 32; n++) { if (Msg.ChEn[n]) { old =
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0]; if (old ==
src1_X[v].data_size[n]) {
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] =
src1_Y[0].data_size[n]; } dest[0].data_size[n] = old; } }
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.

DP_ATOMIC_CMPXCHG - Atomic Compare Exchange

28:25 **Src0 Length**

Format:	DP_ADDR_REG_SIZE

Specifies the size of the address payload, in registers.

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

<table>
<tr><th colspan="3">DP_ATOMIC_CMPXCHG - Atomic Compare Exchange</th></tr>
<tr><td></td><td>14:12</td><td><b>Vector Size</b>
<table>
<tr><td>Default Value:</td><td>0 V1</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>11:9</td><td><b>Data Size</b>
<table>
<tr><td>Format:</td><td>DP_DATA_SIZE</td></tr>
</table>
Specifies both bit size of the data payload item in memory and the bit size used in the register payload.
<table>
<tr><th>Restriction</th><th>Source</th></tr>
<tr><td>Restriction : D64 atomic operations not supported on SLM.</td><td>SFID_E</td></tr>
<tr><td>Restriction : D64 atomic operations are not supported on SFID_D (TGM).</td><td>SFID_D</td></tr>
<tr><td>Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLOAT.</td><td>SFID_D</td></tr>
<tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr>
</table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b>
<table>
<tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr>
</table>
Specifies the bit size of each address payload item.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>5:0</td><td><b>Atomic Operation</b>
<table>
<tr><td>Default Value:</td><td>18 Atomic Compare Exchange</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
</table>

Atomic Decrement

DP_ATOMIC_DEC - Atomic Decrement

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Atomic decrement of memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload is null.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.DEC.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_nullreg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base; for
(n = 0; n < 32; n++) { if (Msg.ChEn[n]) { old =
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = old - 1;
dest[0].data_size[n] = old; } }
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_ATOMIC_DEC - Atomic Decrement

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

<table>
<tr><th colspan="3">DP_ATOMIC_DEC - Atomic Decrement</th></tr>
<tr><td rowspan="4"></td><td>11:9</td><td><b>Data Size</b>
<table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>
Specifies both bit size of the data payload item in memory and the bit size used in the register payload.
<table>
<tr><th>Restriction</th><th>Source</th></tr>
<tr><td>Restriction : D64 atomic operations are not supported on SFID_E (SLM) and SFID_D (TGM).</td><td>SFID_D,
SFID_E</td></tr>
<tr><td>Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLOAT.</td><td>SFID_D</td></tr>
<tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr>
</table></td></tr>
<tr><td>8:7</td><td><b>Address Size</b>
<table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>
Specifies the bit size of each address payload item.</td></tr>
<tr><td>6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>5:0</td><td><b>Atomic Operation</b>
<table><tr><td>Default Value:</td><td>9 Atomic Decrement</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Atomic Float Add

DP_ATOMIC_FADD - Atomic Float Add

Source:	SFID_1, SFID_E, SFID_F
Length Bias:	1

Atomic add of src1 from memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.
Floating point atomic add is not supported for SFID_E (SLM).

Restriction
Floating point atomic add is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.FADD.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base; for
(n = 0; n < 32; n++) { if (Msg.ChEn[n]) { old =
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = old +
src1[0].data_size[n]; dest[0].data_size[n] = old; } }
```

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>30:29</td><td><b>Address Type</b><table><tr><td>Format:</td><td>DP_ADDR_SURFACE_TYPE</td></tr></table>Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc.<table><tr><th>Restriction</th></tr><tr><td>Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER.</td></tr></table></td></tr>
</table>

DP_ATOMIC_FADD - Atomic Float Add

28:25 **Src0 Length**

Format:	DP_ADDR_REG_SIZE

Specifies the size of the address payload, in registers.

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

<table>
<tr><th colspan="3">DP_ATOMIC_FADD - Atomic Float Add</th></tr>
<tr><td></td><td>14:12</td><td><b>Vector Size V1</b><table><tr><td>Default Value:</td><td>0 V1</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>11:9</td><td><b>Data Size</b><table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>Specifies both bit size of the data payload item in memory and the bit size used in the register payload.<table><tr><th>Restriction</th><th>Source</th></tr><tr><td>Data size D64 for float_add is not allowed.</td><td></td></tr><tr><td>Data sizes D8, D8U32, D16, D16U32 for float_add is not allowed.</td><td></td></tr><tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b><table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>Specifies the bit size of each address payload item.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Atomic Operation</b><table><tr><td>Default Value:</td><td>19 Atomic Float Add</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Atomic Float Compare Exchange

DP_ATOMIC_FCMPXCHG - Atomic Float Compare Exchange

Source:	SFID_1, SFID_E, SFID_F
Length Bias:	1

Atomic compare src1_X and memory data and replace if equal with src1_Y. Returns the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size. The src1 data payload is a vector of 2 values (X, Y). X is the value to match, and Y is the value to replace it with.
The dest data payload format is selected by Data Size.

Restriction
Floating point atomics are not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.FCMPXCHG.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
for (n = 0; n < 32; n++) {
    if (Msg.ChEn[n]) {
        old = ((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
        if (old == src1_X[v].data_size[n]) {
            ((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = src1_Y[0].data_size[n];
        }
        dest[0].data_size[n] = old;
    }
}
```

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>30:29</td><td><b>Address Type</b>
<table>
<tr><td>Format:</td><td>DP_ADDR_SURFACE_TYPE</td></tr>
</table>
Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc.
<table>
<tr><th>Restriction</th></tr>
<tr><td>Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.</td></tr>
</table></td></tr>
</table>

DP_ATOMIC_FCMPXCHG - Atomic Float Compare Exchange

28:25 **Src0 Length**

Format:	DP_ADDR_REG_SIZE

Specifies the size of the address payload, in registers.

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

DP_ATOMIC_FCMPXCHG - Atomic Float Compare Exchange

<table>
<tr><td></td><td>14:12</td><td><b>Vector Size V1</b><table><tr><td>Default Value:</td><td>0 V1</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>11:9</td><td><b>Data Size</b><table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>Specifies both bit size of the data payload item in memory and the bit size used in the register payload.<table><tr><th>Restriction</th></tr><tr><td>Data size of D64 is not allowed for this atomic operation.</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b><table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>Specifies the bit size of each address payload item.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Atomic Operation</b><table><tr><td>Default Value:</td><td>23 Atomic Float Compare Exchange</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Atomic Float Max

DP_ATOMIC_FMAX - Atomic Float Max

Source:	SFID_1, SFID_E, SFID_F
Length Bias:	1

Atomic store the max of src1 and memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.

Restriction
Floating point atomics are not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.FMAX.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
for (n = 0; n < 32; n++) {
    if (Msg.ChEn[n]) {
        old = ((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
        ((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = fmax(old, src1[0].data_size[n]);
        dest[0].data_size[n] = old;
    }
}
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_ATOMIC_FMAX - Atomic Float Max

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

DP_ATOMIC_FMAX - Atomic Float Max

	11:9	**Data Size** Format: DP_DATA_SIZE Specifies both bit size of the data payload item in memory and the bit size used in the register payload. **Restriction** Data size of D64 is not allowed for this atomic operation.
	8:7	**Address Size** Format: DP_ADDR_SIZE Specifies the bit size of each address payload item.
	6	**Reserved** Access: RO Format: MBZ
	5:0	**Atomic Operation** Default Value: 22 Atomic Float Max Format: Opcode

Atomic Float Min

DP_ATOMIC_FMIN - Atomic Float Min

Source:	SFID_1, SFID_E, SFID_F
Length Bias:	1

Atomic store the min of src1 and memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.

Restriction
Floating point atomics are not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.FMIN.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
for (n = 0; n < 32; n++) {
    if (Msg.ChEn[n]) {
        old = ((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
        ((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = fmin(old, src1[0].data_size[n]);
        dest[0].data_size[n] = old;
    }
}
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Programming Notes** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

DP_ATOMIC_FMIN - Atomic Float Min

	11:9	**Data Size** Format: DP_DATA_SIZE Specifies both bit size of the data payload item in memory and the bit size used in the register payload. **Restriction** Data size of D64 is not allowed for this atomic operation.
	8:7	**Address Size** Format: DP_ADDR_SIZE Specifies the bit size of each address payload item.
	6	**Reserved** Access: RO Format: MBZ
	5:0	**Atomic Operation** Default Value: 21 Atomic Float Min Format: Opcode

Atomic Float Sub

DP_ATOMIC_FSUB - Atomic Float Sub

Source:	SFID_1, SFID_E, SFID_F
Length Bias:	1

Atomic subtract of src1 from memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.

Restriction
Floating point atomic sub is not supported for SFID_D (TGM).
Floating point atomic sub is not supported for SFID_E (SLM)

Syntax

```
[(pred)] ATOMIC.FSUB.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base for
(n = 0; n < 32; n++) { if (Msg.ChEn[n]) { old =
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = old -
src1[0].data_size[n]; dest[0].data_size[n] = old; } }
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_ATOMIC_FSUB - Atomic Float Sub

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

DP_ATOMIC_FSUB - Atomic Float Sub

<table>
<tr><td></td><td>11:9</td><td><b>Data Size</b>
<table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>
Specifies both bit size of the data payload item in memory and the bit size used in the register payload.
<table>
<tr><th>Restriction</th><th>Source</th></tr>
<tr><td>D64 float_sub is not supported.</td><td></td></tr>
<tr><td>Data sizes D8, D8U32, D16, D16U32 for float_sub is not allowed.</td><td></td></tr>
<tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr>
</table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b>
<table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>
Specifies the bit size of each address payload item.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Atomic Operation</b>
<table><tr><td>Default Value:</td><td>20 Atomic Float Sub</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Atomic Increment

DP_ATOMIC_INC - Atomic Increment

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Atomic increment of memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload is null.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.INC.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_nullreg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
for (n = 0; n < 32; n++) {
    if (Msg.ChEn[n]) {
        old = ((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
        ((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = old + 1;
        dest[0].data_size[n] = old;
    }
}
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_ATOMIC_INC - Atomic Increment

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

DP_ATOMIC_INC - Atomic Increment

<table>
<tr><td></td><td>11:9</td><td><b>Data Size</b>
<table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>Specifies both bit size of the data payload item in memory and the bit size used in the register payload.<table><tr><th>Restriction</th><th>Source</th></tr><tr><td>Restriction : D64 atomic operations are not supported on SFID_E (SLM) and SFID_D (TGM).</td><td>SFID_D, SFID_E</td></tr><tr><td>Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLOAT.</td><td>SFID_D</td></tr><tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b>
<table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>Specifies the bit size of each address payload item.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Atomic Operation</b>
<table><tr><td>Default Value:</td><td>8 Atomic Increment</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Atomic Load

DP_ATOMIC_LOAD - Atomic Load

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Atomic read of the memory data value, without modifying the data. For each enabled SIMT lane, a scalar value is returned.

Programming Notes
The operation differs from load operations because it is sequentially ordered with other atomic operations and follows atomic operation cache policies.
The src0 address payload format is selected by Address Size.
The src1 data payload is null.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.LD.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_nullreg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
for (n = 0; n < 32; n++) {
    if (Msg.ChEn[n]) {
        dest[0].data_size[n] = ((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
    }
}
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: **DP_ADDR_SURFACE_TYPE** Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: **DP_ADDR_REG_SIZE** Specifies the size of the address payload, in registers.

DP_ATOMIC_LOAD - Atomic Load

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

11:9 **Data Size**

Format:	DP_DATA_SIZE

Specifies both bit size of the data payload item in memory and the bit size used in the register payload.

DP_ATOMIC_LOAD - Atomic Load

Restriction	Source
Restriction : D64 atomic operations are not supported on SFID_E (SLM) and SFID_D (TGM).	SFID_D, SFID_E
Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLOAT.	SFID_D
For SFID_1 (UGML), data size D64 is only supported with address size A64.	SFID_1

8:7 **Address Size**

Format:	**DP_ADDR_SIZE**

Specifies the bit size of each address payload item.

6 **Reserved**

Access:	RO
Format:	MBZ

5:0 **Atomic Operation**

Default Value:	10 Atomic Load
Format:	Opcode

Atomic Max

DP_ATOMIC_MAX - Atomic Max

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Atomic store the signed int max of src1 and memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.MAX.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
 for (n = 0; n < 32; n++) { if (Msg.ChEn[n]) { old =
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = signed_max(old,
src1[0].data_size[n]); dest[0].data_size[n] = old; } }
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_ATOMIC_MAX - Atomic Max

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

<table>
<tr><th colspan="3">DP_ATOMIC_MAX - Atomic Max</th></tr>
<tr><td></td><td>11:9</td><td><b>Data Size</b>
<table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>
Specifies both bit size of the data payload item in memory and the bit size used in the register payload.
<table>
<tr><th>Restriction</th><th>Source</th></tr>
<tr><td>Restriction : D64 atomic operations are not supported on SFID_E (SLM) and SFID_D (TGM).</td><td>SFID_D, SFID_E</td></tr>
<tr><td>Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLOAT.</td><td>SFID_D</td></tr>
<tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr>
</table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b>
<table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>
Specifies the bit size of each address payload item.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Atomic Operation</b>
<table><tr><td>Default Value:</td><td>15 Atomic Max</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Atomic Min

DP_ATOMIC_MIN - Atomic Min

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Atomic store the signed int min of src1 and memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.MIN.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
 for (n = 0; n < 32; n++) { if (Msg.ChEn[n]) { old =
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = signed_min(old,
src1[0].data_size[n]); dest[0].data_size[n] = old; } }
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_ATOMIC_MIN - Atomic Min

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16.
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

DP_ATOMIC_MIN - Atomic Min

<table>
<tr><td></td><td>11:9</td><td><b>Data Size</b>
<table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>
Specifies both bit size of the data payload item in memory and the bit size used in the register payload.
<table>
<tr><th>Restriction</th><th>Source</th></tr>
<tr><td>Restriction : D64 atomic operations are not supported on SFID_E (SLM) and SFID_D (TGM).</td><td>SFID_D,
SFID_E</td></tr>
<tr><td>Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLOAT.</td><td>SFID_D</td></tr>
<tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr>
</table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b>
<table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>
Specifies the bit size of each address payload item.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Atomic Operation</b>
<table><tr><td>Default Value:</td><td>14 Atomic Min</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Atomic OR

DP_ATOMIC_OR - Atomic OR

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Atomic store the bitwise OR of src1 and memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.OR.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
 for (n = 0; n < 32; n++) { if (Msg.ChEn[n]) { old =
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = old |
src1[0].data_size[n]; dest[0].data_size[n] = old; } }
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_ATOMIC_OR - Atomic OR

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

<table>
<tr><th colspan="3">DP_ATOMIC_OR - Atomic OR</th></tr>
<tr><td rowspan="4"></td><td>11:9</td><td><b>Data Size</b>
<table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>
Specifies both bit size of the data payload item in memory and the bit size used in the register payload.
<table>
<tr><th>Restriction</th><th>Source</th></tr>
<tr><td>Restriction : D64 atomic operations are not supported on SFID_E (SLM) and SFID_D (TGM).</td><td>SFID_D, SFID_E</td></tr>
<tr><td>Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLOAT.</td><td>SFID_D</td></tr>
<tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr>
</table></td></tr>
<tr><td>8:7</td><td><b>Address Size</b>
<table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>
Specifies the bit size of each address payload item.</td></tr>
<tr><td>6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>5:0</td><td><b>Atomic Operation</b>
<table><tr><td>Default Value:</td><td>25 Atomic OR</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Atomic Store

DP_ATOMIC_STORE - Atomic Store

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Store untyped data to memory. For each enabled SIMT lane, a scalar is written into memory from registers.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.STORE.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
 for (n = 0; n < 32; n++) { if (Msg.ChEn[n]) { old =
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] =
src1[0].data_size[n]; dest[0].data_size[n] = old; } }
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: **DP_ADDR_SURFACE_TYPE** Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Programming Notes** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: **DP_ADDR_REG_SIZE** Specifies the size of the address payload, in registers.

DP_ATOMIC_STORE - Atomic Store

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

11:9 **Data Size**

Format:	DP_DATA_SIZE

Specifies both bit size of the data payload item in memory and the bit size used in the register payload.

DP_ATOMIC_STORE - Atomic Store

<table>
<tr><td></td><td></td><td><table><tr><th>Restriction</th><th>Source</th></tr><tr><td>Restriction : D64 atomic operations are not supported on SFID_E (SLM) and SFID_D (TGM).</td><td>SFID_D,
SFID_E</td></tr><tr><td>Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLOAT.</td><td>SFID_D</td></tr><tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b><table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>Specifies the bit size of each address payload item.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Atomic Operation</b><table><tr><td>Default Value:</td><td>11 Atomic Store</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Atomic Sub

DP_ATOMIC_SUB - Atomic Sub

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Atomic signed int subtract of src1 from memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.SUB.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
 for (n = 0; n < 32; n++) { if (Msg.ChEn[n]) { old =
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = old -
src1[0].data_size[n]; dest[0].data_size[n] = old; } }
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_ATOMIC_SUB - Atomic Sub

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

<table>
<tr><th colspan="3">DP_ATOMIC_SUB - Atomic Sub</th></tr>
<tr><td rowspan="4"></td><td>11:9</td><td><b>Data Size</b>
<table>
<tr><td>Format:</td><td>DP_DATA_SIZE</td></tr>
</table>
Specifies both bit size of the data payload item in memory and the bit size used in the register payload.
<table>
<tr><th>Restriction</th><th>Source</th></tr>
<tr><td>Restriction : D64 atomic operations are not supported on SFID_E (SLM) and SFID_D (TGM).</td><td>SFID_D,
SFID_E</td></tr>
<tr><td>Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLT.</td><td>SFID_D</td></tr>
<tr><td>Data size D64 for float_sub is not allowed.</td><td></td></tr>
<tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr>
</table>
</td></tr>
<tr><td>8:7</td><td><b>Address Size</b>
<table>
<tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr>
</table>
Specifies the bit size of each address payload item.
</td></tr>
<tr><td>6</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>5:0</td><td><b>Atomic Operation</b>
<table>
<tr><td>Default Value:</td><td>13 Atomic Sub</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table>
</td></tr>
</table>

Atomic UMax

DP_ATOMIC_UMAX - Atomic UMax

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Atomic store the unsigned int max of src1 and memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.UMAX.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
 for (n = 0; n < 32; n++) { if (Msg.ChEn[n]) { old =
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = unsigned_max(old,
src1[0].data_size[n]); dest[0].data_size[n] = old; } }
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_ATOMIC_UMAX - Atomic UMax

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

<table>
<tr><th colspan="3">DP_ATOMIC_UMAX - Atomic UMax</th></tr>
<tr><td></td><td>11:9</td><td><b>Data Size</b>
<table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>
Specifies both bit size of the data payload item in memory and the bit size used in the register payload.
<table>
<tr><th>Restriction</th><th>Source</th></tr>
<tr><td>Restriction : D64 atomic operations are not supported on SFID_E (SLM) and SFID_D (TGM).</td><td>SFID_D,
SFID_E</td></tr>
<tr><td>Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLOAT.</td><td>SFID_D</td></tr>
<tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr>
</table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b>
<table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>
Specifies the bit size of each address payload item.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Atomic Operation</b>
<table><tr><td>Default Value:</td><td>17 Atomic UMax</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Atomic UMin

DP_ATOMIC_UMIN - Atomic UMin

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Atomic store the unsigned int min of src1 and memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.UMIN.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
for (n = 0; n < 32; n++) { if (Msg.ChEn[n]) { old =
((Base+offset)+(src0.addr_size[n])).data_size[0];
((Base+offset)+(src0.addr_size[n])).data_size[0] = unsigned_min(old,
src1[0].data_size[n]); dest[0].data_size[n] = old; } }
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_ATOMIC_UMIN - Atomic UMin

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

<table>
<tr><th colspan="3">DP_ATOMIC_UMIN - Atomic UMin</th></tr>
<tr><td></td><td>11:9</td><td><b>Data Size</b>
<table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>
Specifies both bit size of the data payload item in memory and the bit size used in the register payload.
<table>
<tr><th>Restriction</th><th>Source</th></tr>
<tr><td>Restriction : D64 atomic operations are not supported on SFID_E (SLM) and SFID_D (TGM).</td><td>SFID_D,
SFID_E</td></tr>
<tr><td>Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLOAT.</td><td>SFID_D</td></tr>
<tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr>
</table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b>
<table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>
Specifies the bit size of each address payload item.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Atomic Operation</b>
<table><tr><td>Default Value:</td><td>16 Atomic UMin</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Atomic XOR

DP_ATOMIC_XOR - Atomic XOR

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Atomic store the bitwise XOR of src1 and memory data and return the old value. For each enabled SIMT lane, a scalar is written into memory.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size.
The dest data payload format is selected by Data Size.

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] ATOMIC.XOR.sfid[.cache] (exec_mask) dest_reg:data_size
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
 for (n = 0; n < 32; n++) { if (Msg.ChEn[n]) { old =
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0];
((msg_base_address+offset)+(src0.addr_size[n])).data_size[0] = old ^
src1[0].data_size[n]; dest[0].data_size[n] = old; } }
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful atomic messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful atomic messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Atomic messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_ATOMIC_XOR - Atomic XOR

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-4		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16
0		No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

Programming Notes	Source
Atomic messages are always forced to "un-cacheable" in the L1 cache.	SFID_1, SFID_D, SFID_F

16 **Reserved**

Access:	RO
Format:	MBZ

15 **No Transpose**

Default Value:	0 SIMT
Format:	Opcode

14:12 **Vector Size V1**

Default Value:	0 V1
Format:	Opcode

<table>
<tr><th colspan="3">DP_ATOMIC_XOR - Atomic XOR</th></tr>
<tr><td></td><td>11:9</td><td><b>Data Size</b>
<table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>
Specifies both bit size of the data payload item in memory and the bit size used in the register payload.
<table>
<tr><th>Restriction</th><th>Source</th></tr>
<tr><td>Restriction : D64 atomic operations are not supported on SFID_E (SLM) and SFID_D (TGM).</td><td>SFID_D, SFID_E</td></tr>
<tr><td>Restriction : Typed atomics require the surface format be one component and match the data size. For example, D32 is valid for surface format R32_UINT, R32_SINT or R32_FLOAT.</td><td>SFID_D</td></tr>
<tr><td>For SFID_1 (UGML), data size D64 is only supported with address size A64.</td><td>SFID_1</td></tr>
</table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b>
<table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>
Specifies the bit size of each address payload item.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Atomic Operation</b>
<table><tr><td>Default Value:</td><td>26 Atomic XOR</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Average

avg - Average

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

The avg instruction performs component-wise integer average of src0 and src1 and stores the results in dst. An integer average uses integer upward rounding. It is equivalent to increment one to the addition of src0 and src1 and then apply an arithmetic right shift to this intermediate value.

Format:

```
      The avg instruction performs component-wise integer average of src0 and src1 and
stores the results in dst. An integer average uses integer upward rounding. It is
equivalent to increment one to the addition of src0 and src1 and then apply an arithmetic
right shift to this intermediate value.
```

Syntax

```
[(pred)] avg[.cmod] (exec_size) reg reg reg
 [(pred)] avg[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = (src0.chan[n] + src1.chan[n] + 1) » 1;// Use arithmetic shift
right.
     }
 }
```

Src Types	Dst Types
*B,*W,*D	*B,*W,*D

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)
	125:122	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ

avg - Average

121:120	**Src1.Mod**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**SrcMod**
119:116	**Src1.VertStride**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**VertStride**
115:113	**Src1.Width**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**Width**
112	**Src1.AddrMode**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**AddrMode**
111:98	**Src1.Operand**		
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
		Format:	**IndirectOperand**
111:98	**Src1.Operand**		
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
		Format:	**DirectOperand**
97:96	**Src1.HorzStride**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**HorzStride**
95:92	**CondCtrl**		
		Format:	**FlagModifier**
91:88	**Src1.DataType**		
		Exists If:	([Src1.IsImm]==true)
		Format:	**ImmDataType**
91:88	**Src1.DataType**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**RegDataType**
87:84	**Src0.VertStride**		
		Format:	**VertStride**
83:81	**Src0.Width**		
		Format:	**Width**
80	**Src0.AddrMode**		
		Format:	**AddrMode**

avg - Average

	Bits		
	79:66	**Src0.Operand**	
		Exists If:	([Src0.AddrMode]==Direct)
		Format:	**DirectOperand**
	79:66	**Src0.Operand**	
		Exists If:	([Src0.AddrMode]==Indirect)
		Format:	**IndirectOperand**
	65:64	**Src0.HorzStride**	
		Format:	**HorzStride**
	63:50	**Dst.Operand**	
		Exists If:	([Dst.AddrMode]==Direct)
		Format:	**DirectOperand**
	63:50	**Dst.Operand**	
		Exists If:	([Dst.AddrMode]==Indirect)
		Format:	**IndirectOperand**
	49:48	**Dst.HorzStride**	
		Format:	**HorzStride**
	47	**Src1.IsImm** This field indicate that Source 1 operand is carrying an immediate value.	
		Value	**Name**
		0	false **[Default]**
		1	true
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value.	
		Value	**Name**
		0	false **[Default]**
		1	true
	45:44	**Src0.Mod**	
		Format:	**SrcMod**
	43:40	**Src0.DataType**	
		Exists If:	([Src0.IsImm]==false)
		Format:	**RegDataType**
	43:40	**Src0.DataType**	
		Exists If:	([Src0.IsImm]==true)
		Format:	**ImmDataType**
	39:36	**Dst.DataType**	
		Format:	**RegDataType**

<table>
<tr><th colspan="3">avg - Average</th></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr></table></td></tr>
</table>

avg - Average

		1 | Negative | Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.
	27:24	**PredCtrl** Format: | **PredCtrl** This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.
	23	**FlagRegNum[0]** This field specifies bit[0] of the register number for a flag register operand.
	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: | **ChanOff** This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: | **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: | **Header**

AVP_BSD_OBJECT

AVP_BSD_OBJECT

Source:	VideoCS
Length Bias:	2

The AVP Pipeline is selected with the **Media Instruction Opcode "8h"** for all AVP Commands. Each AVP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The AVP_BSD_OBJECT command sends to HW a tile at a time from an AV1 bitstream, starting with the first coded byte of the tile, not including the prefixed tile byte size. The bit stream of a tile, tile group, and of a frame may end with trailing bits and extra padding zero bytes. The prefixed tile byte size includes all the trailing bits and padding zero bytes at the end of a tile.
Each tile's coded/compressed bitstream is started and ended at a byte boundary.
HW is not required to parse the trailing bits and padding zero bytes. HW can stop processing right after it has completed the decoding of the last block in the tile. Potentially, error checking can be implemented to detect the trailing bits and padding zeros, but is not implemented in this generation of AVP Pipeline.
here can be multiple tiles in an AV1 frame and thus this command can be issued multiple times per frame. A coded frame minimum has at least 1 tile definition, i.e a tile can cover the entire frame, unless the frame size exceeds the max allowed tile size limits in pixels, then the frame must contain more than 1 tile. There is no compressed header in AV1, hence AVP_BSD_OBJECT command is only used to process the bitstream of each individual tile of a frame.
The AVP_BSD_OBJECT command must be the last command issued in the sequence of batch commands before the AVP Pipeline starts decoding. Prior to issuing this command, it is assumed that all configuration parameters needed by the AVP Pipeline have been loaded in a specific order, including workload configuration registers and configuration tables. When this command is issued, the AVP Pipeline is waiting for bitstream data to be presented to its bitstream input shift register.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 3h Codec/Engine Name Format: OpCode Codec/Engine Name = AV1 = 3h
	22:16	**Media Instruction Command** Default Value: 20h AVP_BSD_OBJECT_STATE Format: OpCode

<table>
<tr><th colspan="3">AVP_BSD_OBJECT</th></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>Dword Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>(Excludes Dwords 0, 1).<table><tr><th>Value</th><th>Name</th></tr><tr><td>1h</td><td></td></tr></table></td></tr>
<tr><td>1</td><td>31:0</td><td><b>Tile Indirect BSD Data Length</b><table><tr><td>Format:</td><td>U32</td></tr></table>It specifies the compressed bitstream byte size of a tile.
Each tile is started and ended at a byte boundary.
It has the same value as the prefixed tile byte size read from the bitstream at the beginning of a tile. The Data Length does not include this prefixed tile byte size, but does include any zero padding bytes at the end of a tile.
Error Checking :
1) only when tile bitstream has run out when the last block of the current tile has not been decoded. Set the error bit in MMIO, and perform error concealment to fill in the missing decoded block(s).
2) there is no checking when there are bytes remaining after the last block of the current tile has been decoded.
Note : HW AVP decoding pipeline is not required to parse the trailing bits and padding zeros at the end of a tile, tile group, and of a frame.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>[1,4294967295]</td><td>Tile_Bitstream_Data_Length</td><td>It has a valid range of 1 to 4294967295 (2^32-1) bytes. Zero byte length is not allowed.
2^31 should be sufficient to handle a 16Kx16K,12-bit,4:4:4 frame size with a compression ratio of 1.
Note : different bitstream LEVELs are having different requirement on the minimum compression ratio.</td></tr></table></td></tr>
<tr><td>2</td><td>31:0</td><td><b>Tile Indirect Data Start Address</b><table><tr><td>Format:</td><td>U32</td></tr></table>Specifies the byte-aligned graphics memory starting address of a tile in the bitstream of a frame relative to the <b>BSD Indirect Object Base Address</b>.
Each tile's coded bitstream is started and ended at a byte boundary.
A frame with multiple tiles is coded as one bitstream. Indirect Data Start Address for each tile is equivalentto an address offseted from the starting address of the frame bitstream.</td></tr>
</table>

AVP_IND_OBJ_BASE_ADDR_STATE

AVP_IND_OBJ_BASE_ADDR_STATE

Source:	VideoCS
Length Bias:	2

The AVP Pipeline is selected with the **Media Instruction Opcode "8h"** for all AVP Commands. Each AVP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The AVP_IND_OBJ_BASE_ADDR_STATE command is used to define the indirect object base address of the AV1 compressed bitstream in graphics memory. This is a frame level command issued in both encoding and decoding processes.
Although a frame is coded as separate tiles, all tiles' compressed bitstream are still required to line up sequentially as one AV1 bitstream. Hence, there is only one Indirect Object Base Address for the entire AV1 coded frame. If the frame contains more than 1 tiles, the BSD Object Command will be issued multiple times, once for each tile and with its own tile bitstream starting memory address.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 3h Codec/Engine Name Format: OpCode Codec/Engine Name = AVP = 3h
	22:16	**Media Instruction Command** Default Value: 3h AVP_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value / Name: 04h /
1..2	63:0	**AVP Indirect Bitstream Object Base Address** Format: SplitBaseAddress4KByteAligned

AVP_IND_OBJ_BASE_ADDR_STATE

		Description Decoder: Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the AVP_BSD_OBJECT command for reading each tile's compressed bitstream in a frame. Encoder: Specifies memory address for writing each tile's compressed bitstream. There is no specific base address in the PIPE_BUF_ADDR_STATE to add this address on top of it. This address should be programmed for each tile.
3	31:0	**AVP Indirect Bitstream Object Memory Address Attributes** Format: **MemoryAddressAttributes**
4..5	63:0	**Reserved** Access: RO Format: MBZ
6..7	63:0	**AVP Indirect CU Object Base Address** Format: **SplitBaseAddress4KByteAligned** Specifies the 4K-byte aligned data buffer base address for the read-only indirect data object for reading per CU data during the encoding process. Encoder Only
8	31:0	**AVP Indirect CU Object Object Memory Address Attributes** Format: **MemoryAddressAttributes**

AVP_INLOOP_FILTER_STATE

AVP_INLOOP_FILTER_STATE

Source:	VideoCS
Length Bias:	2

The AVP_INLOOP_FILTER_STATE command provides all the frame level syntax elements and derived parameters that are needed for the processing of all the post in-loop filters present in the AV1 codec, except the Luma and Chroma x0_qn which are tile based derived parameters. This includes the Deblocker, the CDEF (Constrained Directional Enhancement Filter), the HSRF (Horizontal-only Super-Resolution Filter), and the LRF (Loop Restoration Filter). These syntax elements can be changed in the bitstream from frame to frame.

All Post In-Loop Filters are inherently frame-based filtering, but when implemented in a HW pipeline, the filtering process is performed in tile based and in a block-by-block fashion. In the addition to these frame and tile level states, there are additional syntax elements and derived parameters that are generated at SuperBlock level, and are not described here.

Each of these 4 Post In-Loop Filters can be controlled independently and each can be enabled or disabled independently. Except the HSRF, all the other 3 filters have separate controls for each color plane as well. To disable a Post In-Loop Filter, its control parameter(s) are set to 0 - the default state.

This command should be issued once per tile, even if no post in-loop filter is enabled for decoding the current frame. When in frame lossless mode or when IntraBC is enabled, all the Post In-Loop Filters are disabled for all color planes, this command will provide the default values for all parameters. All syntax elements are then assumed a value of 0, except otherwise specified in each field of this State Command.

When it is in monochrome video, no filter parameter for the two chroma planes is present in the bitstream.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: Opcode
	28:27	**Pipeline Type** Default Value: 2h Format: Opcode
	26:23	**Media Instruction Opcode** Default Value: 3h Codec/Engine Name Format: OpCode Codec/Engine Name = AV1 = 3h
	22:16	**Media Instruction Command** Default Value: 33h AVP_INLOOP_FILTER_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">AVP_INLOOP_FILTER_STATE</th></tr>
<tr><td></td><td>11:0</td><td><b>Dword Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>(Excludes Dwords 0, 1).<table><tr><th>Value</th><th>Name</th></tr><tr><td>Dh</td><td></td></tr></table></td></tr>
<tr><td>1</td><td>31</td><td><b>Deblocker Filter Delta LF Present Flag</b><table><tr><td>Format:</td><td>U1</td></tr></table>set to 1, specifies that additional loop filter delta values are present at the superblock level in the bitstream.
set to 0, specifies that no additional loop filter delta values are present at the superblock level in the bitstream.
It is the frame level syntax element delta_if_present flag. It is present in the bitstream, only if delta_q_present is set to 1. If delta_q_present is set to 0, delta_lf_present flag is not present, and is defaulted to 0.</td></tr>
<tr><td></td><td>30</td><td><b>Deblocker Filter Delta LF Multi Flag</b><table><tr><td>Format:</td><td>U1</td></tr></table>set to 1, specifies that at the Superblock level, separate loop filter deltas (mulitple deltas)are sent for
1) horizontal Luma Y edges,
2) vertical Luma edges,
3) both horizontal and vertical Chroma U edges, and
4) for both horizontal and vertical Chroma V edges.
set to 0, specifies that at the Superblock level, the same loop filter delta is used for all edges of all color planes (Y, U and V).
It is the frame level syntax delta_lf_multi.
It is present in the bitstream, when delta_lf_present flag is set to 1. If delta_lf_present flag is set to 0,delta_lf_multi flag is not present, and is defaulted to 0.</td></tr>
<tr><td></td><td>29:28</td><td><b>Deblocker Delta LF Resolution</b><table><tr><td>Format:</td><td>U2</td></tr></table>It specifies the number of left shift which should be applied to decoded loop filter delta values.
It is in the range of [0..3]. (no shft, and therefore no scaling).
It is theframe level syntax element delta_lf_res.
It is present in the bitstream, when delta_lf_present flag is set to 1. If delta_lf_present flag is set to 0,delta_lf_resis not present, and is defaulted to 0.
Note : it is used to derive one part of the final filter levels: lvl +=read_delta_lflevel() * (1 « delta_lf_res).
Note :But in the reference C model, the same name is used for the derived parameter: delta_lf_res = 1 « (delta_lf_res), which can take a 4-bit of value[1, 2, 4 or 8] instead.</td></tr>
<tr><td></td><td>27</td><td><b>Deblocker Filter Mode Ref Delta Enable Flag</b><table><tr><td>Format:</td><td>U1</td></tr></table>set to 1, means that the filter levels depend on the mode and reference frame used to predict a block.
set to 0, means that the filter level does not depend on the mode and reference frame. Default is</td></tr>
</table>

AVP_INLOOP_FILTER_STATE		
		0. It is the frame level syntax elementloop_filter_delta_enabled, or named asmode_ref_delta_enabled.
	26:24	**Deblocker Filter Sharpness Level** Format: U3 It specifies the sharpness level of the deblocker. It is used to compute the deblocker filter limits (lim and mblim) for each of the 64 possible values of a filter level. The deblocker filter levels and the deblock filter sharpness together determine when a block edge is filtered, and by how much the filtering can change the sample values. It is the frame level syntax elementloop_filter_sharpness, or named as sharpness_level. It is in the range of [0..7]. Default is 0.
	23:18	**Chroma V Deblocker Filter Level** Format: U6 It specifies the deblocker filter strength for both the horizontal (using vertical filters) and the vertical (using horizontal filters) edges of the Chroma Vplane. It is the frame level syntax element loop_filter_level[3], or named as filter_level_v.It is present, [only if filter_level[0] and filter_level[1] are not both set to 0 AND only if the current frame is not a monochrome]. If not present, it is default to 0. It is in the range of [0..63]. Default is 0.
	17:12	**Chroma U Deblocker Filter Level** Format: U6 It specifies the deblocker filter strength for both the horizontal (using vertical filters) and the vertical (using horizontal filters) edges of the Chroma U plane. It is the frame level syntax element loop_filter_level[2], or named as filter_level_u. It is present, [only if filter_level[0] and filter_level[1] are not both set to 0 AND only if the current frame is not a monochrome].If not present, it is default to 0. It is in the range of [0..63]. Default is 0.
	11:6	**Luma Y Deblocker Filter Level Horizontal** Format: U6 It specifies the deblocker filter strength for the horizontal edges (using vertical filters) of the Luma plane. It is the frame syntax element loop_filter_level[1]. It is in the range of [0..63]. Default is 0. Settingloop_filter_level[0] =loop_filter_level[1] = 0, disables the deblocker filter.
	5:0	**Luma Y Deblocker Filter Level Vertical** Format: U6 It specifies the deblocker filter strength for the vertical edges (using horizontal filters) of the Luma plane. It is the frame syntax element loop_filter_level[0]. It is in the range of [0..63]. Default is 0. Settingloop_filter_level[0] =loop_filter_level[1] = 0, disables the deblocker filter.

AVP_INLOOP_FILTER_STATE		
2	31	**Reserved** Access: RO Format: MBZ
	30:24	**Deblocker Filter Ref Deltas[3]** Format: S6 It is in 7-bits 2's complement within the range of [-64to +63]. Initialize to 0 bysetup_past_independence(). If this syntax element is not present in the bitstream (i.e. update_ref_delta=0), Deblocker Filter Ref Deltas[Ref=0 to 7] maintains its value from previous frame. Refer to the full description in Deblocker Filter Ref Deltas[0].
	23	**Reserved** Access: RO Format: MBZ
	22:16	**Deblocker Filter Ref Deltas[2]** Format: S6 It is in 7-bits 2's complement within the range of [-64to +63]. Initialize to 0 bysetup_past_independence(). If this syntax element is not present in the bitstream (i.e. update_ref_delta=0), Deblocker Filter Ref Deltas[Ref=0 to 7] maintains its value from previous frame. Refer to the full description in Deblocker Filter Ref Deltas[0].
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Deblocker Filter Ref Deltas[1]** Format: S6 It is in 7-bits 2's complement within the range of [-64to +63]. Initialize to 0 by setup_past_independence(). If this syntax element is not present in the bitstream (i.e. update_ref_delta=0), Deblocker Filter Ref Deltas[Ref=0 to 7] maintains its value from previous frame. Refer to the full description in Deblocker Filter Ref Deltas[0].
	7	**Reserved** Access: RO Format: MBZ
	6:0	**Deblocker Filter Ref Deltas[0]** Format: S6 It specifies the adjustment needed for the deblocker filter level based on which one of the 8 possible reference frames is in used - Deblocker Filter Ref Deltas[Ref=0 to 7]. Ref=0 to 7 is defined as follows: [0] = INTRA_FRAME [1] = LAST_FRAME [2] = LAST2_FRAME

AVP_INLOOP_FILTER_STATE

		[3] = LAST3_FRAME [4] = GOLDEN_FRAME [5] = BWDREF_FRAME [6] = ALTREF2_FRAME [7] = ALTREF_FRAME Deblocker Filter Ref Deltas[] is used to pre-compute the final deblocker filter level for all blocks within a segment, lvl[seg_id][horz_or_vert][ref_frame[0]][block_coding_mode]. The pre-compute can take place either at the frame header (if delta_lf_present_flag ==0), or at the block level(if delta_lf_present_flag ==0); but the equations and clamping used are different in these 2 cases. It is the frame level syntax element loop_filter_ref_deltas[Ref=0 to 7], or named as ref_deltas[Ref=0 to 7]. It is in 7-bits 2's complement within the range of [-64to +63]. Initialize to 1bysetup_past_independence(). If this syntax element is not present in the bitstream (i.e. update_ref_delta=0), Deblocker Filter Ref Deltas[Ref=0 to 7] maintains its value from previous frame. Note : The 8 elements of the Deblock Filter Ref Deltas[Ref=0 to 7] are initialized differently inside the setup_past_independence(). Deblock Filter Ref Deltas[INTRA_FRAME] is set equal to 1. Deblock Filter Ref Deltas[LAST_FRAME] is set equal to 0. Deblock Filter Ref Deltas[LAST2_FRAME] is set equal to 0. Deblock Filter Ref Deltas[LAST3_FRAME] is set equal to 0. Deblock Filter Ref Deltas[BWDREF_FRAME] is set equal to 0. Deblock Filter Ref Deltas[GOLDEN_FRAME] is set equal to -1. Deblock Filter Ref Deltas[ALTREF_FRAME] is set equal to -1. Deblock Filter Ref Deltas[ALTREF2_FRAME] is set equal to -1.
3	31	**Reserved** Access: RO Format: MBZ
	30:24	**Deblocker Filter Ref Deltas[7]** Format: S6 It is in 7-bits 2's complement within the range of [-64to +63]. Initialize to -1bysetup_past_independence(). If this syntax element is not present in the bitstream (i.e. update_ref_delta=0), Deblocker Filter Ref Deltas[Ref=0 to 7] maintains its value from previous frame. Refer to the full description in Deblocker Filter Ref Deltas[0].
	23	**Reserved** Access: RO Format: MBZ
	22:16	**Deblocker Filter Ref Deltas[6]** Format: S6 It is in 7-bits 2's complement within the range of [-64to +63]. Initialize to -1bysetup_past_independence(). If this syntax element is not present in the bitstream (i.e. update_ref_delta=0), Deblocker Filter Ref Deltas[Ref=0 to 7] maintains its value from previous frame.

AVP_INLOOP_FILTER_STATE

		Refer to the full description in Deblocker Filter Ref Deltas[0].
	15	**Reserved** Access: RO Format: MBZ
	14:8	**Deblocker Filter Ref Deltas[5]** Format: S6 It is in 7-bits 2's complement within the range of [-64to +63]. Initialize to -1bysetup_past_independence(). If this syntax element is not present in the bitstream (i.e. update_ref_delta=0), Deblocker Filter Ref Deltas[Ref=0 to 7] maintains its value from previous frame. Refer to the full description in Deblocker Filter Ref Deltas[0].
	7	**Reserved** Access: RO Format: MBZ
	6:0	**Deblocker Filter Ref Deltas[4]** Format: S6 It is in 7-bits 2's complement within the range of [-64to +63]. Initialize to 0 bysetup_past_independence(). If this syntax element is not present in the bitstream (i.e. update_ref_delta=0), Deblocker Filter Ref Deltas[Ref=0 to 7] maintains its value from previous frame. Refer to the full description in Deblocker Filter Ref Deltas[0].
4	31:15	**Reserved** Access: RO Format: MBZ
	14:8	**Deblocker Filter Mode Deltas[1]** Format: S6 It is in 7-bits 2's complement within the range of [-64to +63]. Initialize to 0 by setup_past_independence(). If this syntax element is not present in the bitstream (i.e. update_mode_delta=0), Deblocker Filter ModeDeltas[BPM=0 to 1] maintains its value from previous frame. Refer to the full description in Deblocker Filter Mode Deltas[0].
	7	**Reserved** Access: RO Format: MBZ
	6:0	**Deblocker Filter Mode Deltas[0]** Format: S6 It specifies the adjustment needed for the deblocker filter level based on which block prediction mode is in used - Deblocker Filter Mode Deltas[BPMode=0 to 1]. BPM=0 to 1is defined by mode_lf_lut[Block Prediction Mode] ; Block Prediction Mode can be one of the 25 possible block prediction modes being defined in AV1.

AVP_INLOOP_FILTER_STATE

		BPM=0, for all 13 intra block prediction modes, for the GLOBALMV inter block prediction mode, and for the GLOBAL_GLOBALMV inter compound block prediction mode. BPM =1, for all the other 10 inter block prediction modes (3 inter block prediction modes and 7 inter compound block prediction modes) Deblocker Filter Mode Deltas[] is used to pre-compute the final deblocker filter level for all blocks within a segment, lvl[seg_id][horz_or_vert][ref_frame[0]][mode_lf_lut[block_prediction_mode]]. The pre-compute can take place either at the frame header (if delta_lf_present_flag ==0), or at the block level(if delta_lf_present_flag ==0); but the equations and clamping used are different in these 2 cases. It is the frame level syntax elementloop_filter_mode_deltas[BPM=0 to 1], or named as mode_deltas[BPM=0 to 1]. It is in 7-bits 2's complement within the range of [-64to +63]. Initialize to 0 bysetup_past_independence(). If this syntax element is not present in the bitstream (i.e. update_mode_delta=0), Deblocker Filter Mode Deltas[BPM=0 to 1] maintains its value from previous frame.
5	31:30	**CDEF Filter Damping Factor Minus3** Format: U2 It specifies the amount of damping in the deringing filter. It is the frame level syntax element, cdef_damping_minus_3. It is also named as cdef_damping (=cdef_damping_minus3+3). It takes on value in the range of [0 to 3]. Default is 0.
	29:28	**CDEF Bits** Format: U2 It is used for 2 purposes: 1) [1 « cdef_bits] specifies the number of frame level CDEF Y and UV strengths to be read from the bitstream in the uncompressed header. 2) cdef_bits specifies the number of bits in the bitstream, at the block level, that need to be read for the CDEF strength to be applied. It is the frame level syntax element, cdef_bits. It is in the range of [0..3]. Default is 0. Note: to disable CDEF Filtering process, set cdef_bits to 0, and set cdef_y_strengths[0]=cdef_uv_strengths[0]=0.
	27:24	**Reserved** Access: RO Format: MBZ
	23:18	**CDEF Y Strength[3]** Format: U6 Refer to the full description in CDEF Y Strength[0].
	17:12	**CDEF Y Strength[2]** Format: U6 Refer to the full description in CDEF Y Strength[0].

AVP_INLOOP_FILTER_STATE

	11:6	**CDEF Y Strength[1]** Format: U6 Refer to the full description in CDEF Y Strength[0].
	5:0	**CDEF Y Strength[0]** Format: U6 It specifies one of the 8 possible filter strengths for the CDEF Luma Y Filter. It is the frame level syntax element cdef_y_strength[i=0 to 7], or also named simply as cdef_strengths[i=0 to 7]. The number of active filter strengths in used for the current frame is equal to [1 « cdef_bits], which can only be [1, 2, 4 or 8]. Each strength is in the range of [0 to 63]. For all filter strengths that are not present in the bitstream, they are defaulted to 0. Note: to disable CDEF Filtering process, set cdef_bits to 0, and set cdef_y_strengths[0]=cdef_uv_strengths[0]=0.
6	31:24	**Reserved** Access: RO Format: MBZ
	23:18	**CDEF Y Strength[7]** Format: U6 Refer to the full description in CDEF Y Strength[0].
	17:12	**CDEF Y Strength[6]** Format: U6 Refer to the full description in CDEF Y Strength[0].
	11:6	**CDEF Y Strength[5]** Format: U6 Refer to the full description in CDEF Y Strength[0].
	5:0	**CDEF Y Strength[4]** Format: U6 Refer to the full description in CDEF Y Strength[0].
7	31:24	**Reserved** Access: RO Format: MBZ
	23:18	**CDEF UV Strength[3]** Format: U6 **Description** Refer to the full description in CDEF UVStrength[0]. In Encoder Mode this value should be equal to CDEF Y Strength[3].

AVP_INLOOP_FILTER_STATE

	17:12	**CDEF UV Strength[2]** Format: U6 **Description** Refer to the full description in CDEF UVStrength[0]. In Encoder Mode this value should be equal to CDEF Y Strength[2].
	11:6	**CDEF UV Strength[1]** Format: U6 **Description** Refer to the full description in CDEF UVStrength[0]. In Encoder Mode this value should be equal to CDEF Y Strength[1].
	5:0	**CDEF UV Strength[0]** Format: U6 **Description** It specifies one of the 8 possible filter strengths for the CDEF Chroma U/VFilter. Chroma U and V share the same strength specification. It is the frame level syntax element cdef_uv_strength[i=0 to 7], or named as cdef_uv_strengths[i=0 to 7]. The number of active filter strengths in used for the current frame is equal to [1 « cdef_bits], which can only be [1, 2, 4 or 8]. For monochrome video, there is no Chroma UV strength in the bitstream, either. Each strength is in the range of [0to 63]. The 7th and 8th-bit are ignored. For all filter strengths that are not present in the bitstream, they are defaulted to 0. Note: to disable CDEF Filtering process, set cdef_bits to 0, and set cdef_y_strengths[0]=cdef_uv_strengths[0]=0. In Encoder Mode this value should be equal to CDEF Y Strength[0]. In Encoder Mode- Y and UV filter strengths can be different
8	31:24	**Reserved** Access: RO Format: MBZ
	23:18	**CDEF UV Strength[7]** Format: U6 **Description** Refer to the full description in CDEF UV Strength[0]. In Encoder Mode this value should be equal to CDEF Y Strength[7].

AVP_INLOOP_FILTER_STATE

	17:12	**CDEF UV Strength[6]** Format: U6 **Description** Refer to the full description in CDEF UVStrength[0]. In Encoder Mode this value should be equal to CDEF Y Strength[6].
	11:6	**CDEF UV Strength[5]** Format: U6 **Description** Refer to the full description in CDEF UV Strength[0]. In Encoder Mode this value should be equal to CDEF Y Strength[5].
	5:0	**CDEF UV Strength[4]** Format: U6 **Description** Refer to the full description in CDEF UV Strength[0]. In Encoder Mode this value should be equal to CDEF Y Strength[4].
9	31:21	**Reserved** Access: RO Format: MBZ
	20:16	**Super-Res Denom** Format: U5 It specifies an integer denominator of a fixed point fractional number (scaling factor) which is used to scale down the current frame width (in pixels) for the subsequent block level decoding process of its bitstream. Super-resolution is only applied to the horizontal direction, so this scaling factor is applied to the frame width only. The numerator of this down-scaling factor is always set to 8. The reduced (scaled down) frame width in pixels = (upscaled frame width in pixels * 8 + (super-res denom»1)) / super-res denom. This division is done by Driver. It is derived from the 3-bits frame level syntax element, coded_denom. Super-res denom = coded_denom + 9, which is always > 8 (i.e. downscaling factor is always < 1.0). That is, the super-resolution down-scaling factor can go from [8/9 to 8/16], and the corresponding up-scaling factor can go from [9/8 to 16/8]. If super-resolution is not enabled, super-res denom is not present in the bitstream, its value is defaulted to 8 (i.e no scaling). Hence, Super-Rest Denom is in the range of [8.. 16]. Value 0 to 7 are not allowed. Different frames in a video sequence can have different super-res denom. Adjacent frames can all have different super-res denoms. Super-res denom is used in multiple functions of the Super-resolution and the Loop Restoration processes, but no fixed-point division is needed.

AVP_INLOOP_FILTER_STATE

		Note : if not monochrome video, the same scaling factor and super-resolution process are applied to the two chroma planes as well; otherwise, only the Luma plane is being processed. Following restrictions apply in Encoder Mode if Wiener Filter enabled: In Single pipe: Only valid values are 10, 12, 14 and 16 In Multi pipe: Only valid value is 16 otherwise all values are valid.
	15:0	**Super-Res Upscaled Frame Width Minus1** Format: U16 It specifies the super-resolution upscaled frame width in pixels, without padding on the right and bottom of the frame. Since, super-resolution is only applied to the horizontal frame width, there is no scaling to the vertical frame height. The input and output of the Loop Restoration Filter and all reference frames in the Display Buffer (DPB) are in Super-Res Upscaled Frame Width. When super-resolution is enabled, the bitstream is coded with the reduced frame width. Hence, the current frame width programmed in the AVP_PIC_STATE is the derived reduced frame width without padding on the right and bottom border of the frame. The scaled downframe width (AVP_PIC_STATE frame width minus1 + 1)= (upscaled frame width * 8 + (super-res denom»1)) / super-res denom. Upscaled frame width is derived as follows: A) In KEY FRAME or INTRA-ONLY NON-KEY FRAME: 1) when frame_size_override_flag is set to 1, it is the frame level syntax element frame_width_minus1. 2) when frame_size_override_flag is set to 1, it is the sequence level syntax element max_frame_width_minus1. B) In INTER FRAME: 1) if the current frame size can be inferred from one of the 7 possible reference frames, it is the derived frame width minus1. Otherwise, 2) when frame_size_override_flag is set to 1, it is the frame level syntax element frame_width_minus1. 3) when frame_size_override_flag is set to 1, it is the sequence level syntax element max_frame_width_minus1. Max frame size is 64K, but intel only support up to 16K. Hence, bit 14 and 15 are not used. Default is 7, for minimum frame width is 8 pixels. Note : if upscaling factor > 1.0, AVP_PIC_STATE Frame Width must < 16K. Note : both the upscaled frame width and the downscaled frame width are provided to the AVP HW pipeline, so that HW does not need to perform the division in the scaling process.
10	31:11	**Reserved** Access: RO Format: MBZ
	10	**Use Same Loop Restoration Unit Size for Chromas UV Flag** Format: U1 It specifies how the Chroma UV Loop Restoration Unit size should be derived from that of the Luma. Chroma U and Chroma V are having the same LRU Size. Set to 1, if Chroma LRU size is the same as that of the Luma.

<table>
<tr><th colspan="3">AVP_INLOOP_FILTER_STATE</th></tr>
<tr><td></td><td></td><td>Set to 0, if Chroma LRU size is subsampled that of the Luma in 4:2:0 by half in each dimension. Default is 0.
It is the frame level syntax element, lr_uv_shift, which is present only if current video is a 4:2:0 and not both Chroma plane having filter type set to RESTORE_NONE.
For 4:2:2 and 4:4:4, LRU size for Chromas UV is always the same as that of Luma.
For monochrome, this field is ignored.
Intel Encoder PAK only support LRU size = SuperBlock size = 64x64 pixels, this field must also be set to 0.</td></tr>
<tr><td></td><td>9:8</td><td><b>Loop Restoration Unit Size for Luma Y</b>
<table><tr><td>Format:</td><td>U2</td></tr></table>
It specifies the Loop Restoration Unit size in pixels for Luma Y.
Loop Restoration Unit is a square grid in the current frame for each color plane. Each Loop Restoration Unit of a color plane has its own set of Loop Restoration Filter parameters.
It is derived from at most 2 1-bit syntax element (lr_unit_shift) in the frame header which are present only if not all 3 color planes having filter type set to RESTORE_NONE. If all 3 color planes having filter type set to RESTORE_NONE, then Loop Restoration Unit Size for Luma plane is defaulted to 256x256.
It is also named as LoopRestorationSize[i=0], or restoration_unit_size for Luma plane. But intel has mapped them to the following code:
0, for 0 size (Default, when LR is not enabled)
1, for 64x64 pixels LRU nominal size.
2, for 128x128 pixels LRU nominal size.
3, for 256x256 pixels LRU nominal size.
Intel Encoder PAK only supportLRU size = SuperBlock size = 64x64 pixels.
Note : LRU Luma Size cannot < Superblock Size. If Superblock Size is 128x128, LRU Luma Size cannot be 64x64.</td></tr>
<tr><td></td><td>7:6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:4</td><td><b>Frame Loop Restoration Filter Type for Chroma V</b>
<table><tr><td>Format:</td><td>U2</td></tr></table>
<table><tr><td>It specifies the Frame Level Loop Restoration Filter Type for the Chroma Vplane.
It is derived from 2 bits syntax element (lr_type) in the frame header, if not in a monochrome video. It is also named as frame restoration type[i=2], or frame_restoration_type for Chroma V plane.
It is in the range of [0..3]. For a monochrome video, it is defaulted to 0.
0, for RESTORE_NONE.
1, for RESTORE_WIENER.
2, for RESTORE_SGRPROJ (Dual Self-Guided Projection Filter).
3, for RESTORE_SWITCHABLE (LRU level choice of NONE, WIENER or SGRPROJ).
Each color plane can have its own Frame Level Loop Restoration Filter Type specification.
Note : to disable Loop Restoration Filtering in the current frame, the Frame Restoration Filter Type for all color planes must be set to RESTORE_NONE.</td></tr></table></td></tr>
</table>

AVP_INLOOP_FILTER_STATE

		Note : that the syntax element lr_type uses a different enum order for the four LR filter types (0 for RESTORE_NONE, 1 for SWITCHABLE, 2 for WIENER, and 3 for SGRPROJ). In Encoder Mode, Loop Restoration Filter is not supported, this field can only be set to RESTORE_NONE.
	3:2	**Frame Loop Restoration Filter Type for Chroma U** Format: U2 It specifies the Frame Level Loop Restoration Filter Type for the Chroma U plane. It is derived from 2 bits syntax element (lr_type) in the frame header, if not in a monochrome video. It is also named as framerestorationtype[i=1], orframe_restoration_type for Chroma U plane. It is in the range of [0..3]. For a monochrome video, it is defaulted to 0. 0, for RESTORE_NONE. 1, for RESTORE_WIENER. 2, for RESTORE_SGRPROJ (Dual Self-Guided Projection Filter). 3, for RESTORE_SWITCHABLE (LRU level choice of NONE, WIENER or SGRPROJ). Each color plane can have its own Frame Level Loop Restoration Filter Type specification. Note : to disable Loop Restoration Filtering in the current frame, the Frame Restoration Filter Type for all color planes must be set to RESTORE_NONE. Note : that the syntax element lr_type uses a different enum order for the four LR filter types (0 for RESTORE_NONE, 1 for SWITCHABLE, 2 for WIENER, and 3 for SGRPROJ). In Encoder Mode, Loop Restoration Filter is not supported, this field can only be set to RESTORE_NONE.
	1:0	**Frame Loop Restoration Filter Type for Luma Y** Format: U2 It specifies the Frame Level Loop Restoration Filter Type for the Luma plane. It is derived from 2 bits syntax element (lr_type) in the frame header. It is also named as framerestorationtype[i=0], or frame_restoration_type for Luma plane. It is in the range of [0..3]. Default is 0. 0, for RESTORE_NONE. 1, for RESTORE_WIENER. 2, for RESTORE_SGRPROJ (Dual Self-Guided Projection Filter). 3, for RESTORE_SWITCHABLE (LRU level choice of NONE, WIENER or SGRPROJ). Each color plane can have its own Frame Level Loop Restoration Filter Type specification. Note : to disable Loop Restoration Filtering in the current frame, the Frame Restoration Filter Type for all color planes must be set to RESTORE_NONE. Note : that the syntax element lr_type uses a different enum order for the four LR filter types (0 for RESTORE_NONE, 1 for SWITCHABLE, 2 for WIENER, and 3 for SGRPROJ). In Encoder Mode, Loop Restoration Filter is not supported, this field can only be set to RESTORE_NONE.
11	31:16	**Reserved (for higher precision of x_step_qn)** Format: MBZ

AVP_INLOOP_FILTER_STATE		
	15:0	**Luma Plane x_step_qn** Format: U16 Derived parameter, specifies the increment of the super-res luma upscaling step in pixel position for the current frame.
12	31:0	**Luma Plane x0_qn** Format: S31 Derived parameters, specifies the starting offset (in 2's complement signed integer) of the super-res upscaling process for each tile in the original frame for Luma.
13	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Chroma Plane x_step_qn** Format: U16 Derived parameter, specifies the increment of the super-res chroma upscaling step in pixel position for the current frame.
14	31:0	**Chroma Plane x0_qn** Format: S31 Derived parameters, specifies the starting offset (in 2's complement signed integer) of the super-res upscaling process for each tile in the original frame for chroma.

AVP_INTER_PRED_STATE

AVP_INTER_PRED_STATE

Source:	VideoCS
Length Bias:	2

The AVP Pipeline is selected with the **Media Instruction Opcode "8h"** for all AVP Commands. Each AVP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.
AVP supports a 8-reference frames display buffer. But at any given frame being decoded, only up to 7reference frames out of the 8 can be active. There are also further constraints on which of these 7frames can be used for forward and backward reference in the compound mode.
To simplify the decoder command sequence programming, this command is issued once for each inter-coded tile as well as for each intra-coded tile (such as in a KEY_FRAME, a DELAY_KEY_FRAME, an INTRA_ONLY_FRAME).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 3h Codec/Engine Name Format: OpCode Codec/Engine Name = AVP = 3h
	22:16	**Media Instruction Command** Default Value: 12h AVP_INTER_PRED_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: Dh — Name:
1	31:24	**Saved Order Hints for All References[0][3]** Format: U8
	23:16	**Saved Order Hints for All References[0][2]** Format: U8

AVP_INTER_PRED_STATE

	15:8	**Saved Order Hints for All References[0][1]** Format: U8
	7:0	**Saved Order Hints for All References[0][0]** Format: U8 This array specifies the saved order hints for all references for the purpose of computing the Motion Field Projections. Array indices are defined as [a reference frame index][the 7 reference indices of the reference frame]. Hence, the array indices' range are [0..6] [0..6]. Index value = 0, is set for LAST frame. Index value = 6 is set for ALTREF frame. Since a current frame can have up to 7 references, when each reference was previously decoded as a current frame, it could also have up to 7 references (references of a reference frame), hence totally max. there can be 7x7=49 references in the past. With each reference has its own order hint (previously read from the bitstream), a total of max. 49 order hints need to be saved and programmed here. Order Hint is acting like a form of time step, and can take on a value of [0..255] (8-bits unsigned). All order hints are syntax elements previously read from the frame headers in the bitstream (not a derived value). Note : for tiles of a KEY_FRAME, saved order hints should be all set to a value of 0.
2	31:24	**Active Reference Bitmask for Motion Field Projection** Format: U8 This field specifies which ones of the 7 references are involved in the Motion Field Projection. bit 0 corresponding to the LAST reference frame for the decoding current frame. bit 6corresponding to the ALTREF reference frame for decoding the current frame. bit 7 is reserved and must set to 0. This is an intel derived parameters. A reference has its corresponding bit in the bitmaskset to 1 if 1) motion_field_projection() is called for that reference from av1_setip_motion_field(); AND 2) motion_field_projection() for that reference does not return 0, meaning the following conditions have to be satisfied for that reference: a) Reference is available (i.e. frame buffer index has a valid value) b) Reference is not intra-only (i.e. neither a KEY FRAME nor an INTRA-ONLY FRAME) c) Reference is not scaled. Otherwise set to 0.
	23:16	**Saved Order Hints for All References[0][6]** Format: U8
	15:8	**Saved Order Hints for All References[0][5]** Format: U8
	7:0	**Saved Order Hints for All References[0][4]** Format: U8
3	31:24	**Saved Order Hints for All References[1][3]** Format: U8

AVP_INTER_PRED_STATE

	23:16	**Saved Order Hints for All References[1][2]** Format: U8
	15:8	**Saved Order Hints for All References[1][1]** Format: U8
	7:0	**Saved Order Hints for All References[1][0]** Format: U8 This array specifies the saved order hints for all references for the purpose of computing the Motion Field Projections. Array indices are defined as [a reference frame index][the 7 reference indices of the reference frame]. Hence, the array indices' range are [0..6] [0..6]. Index value = 0, is set for LAST frame. Index value = 6 is set for ALTREF frame. Since a current frame can have up to 7 references, when each reference was previously decoded as a current frame, it could also have up to 7 references (references of a reference frame), hence totally max. there can be 7x7=49 references in the past. With each reference has its own order hint (previously read from the bitstream), a total of max. 49 order hints need to be saved and programmed here. Order Hint is acting like a form of time step, and can take on a value of [0..255] (8-bits unsigned). All order hints are syntax elements previously read from the frame headers in the bitstream (not a derived value). Note : for tiles of a KEY_FRAME, saved order hints should be all set to a value of 0.
4	31:24	**Reserved** Access: RO Format: MBZ
	23:16	**Saved Order Hints for All References[1][6]** Format: U8
	15:8	**Saved Order Hints for All References[1][5]** Format: U8
	7:0	**Saved Order Hints for All References[1][4]** Format: U8
5	31:24	**Saved Order Hints for All References[2][3]** Format: U8
	23:16	**Saved Order Hints for All References[2][2]** Format: U8
	15:8	**Saved Order Hints for All References[2][1]** Format: U8
	7:0	**Saved Order Hints for All References[2][0]** Format: U8 This array specifies the saved order hints for all references for the purpose of computing the Motion Field Projections. Array indices are defined as [a reference frame index][the 7 reference indices of the reference

AVP_INTER_PRED_STATE

		frame]. Hence, the array indices' range are [0..6] [0..6]. Index value = 0, is set for LAST frame. Index value = 6 is set for ALTREF frame. Since a current frame can have up to 7 references, when each reference was previously decoded as a current frame, it could also have up to 7 references (references of a reference frame), hence totally max. there can be 7x7=49 references in the past. With each reference has its own order hint (previously read from the bitstream), a total of max. 49 order hints need to be saved and programmed here. Order Hint is acting like a form of time step, and can take on a value of [0..255] (8-bits unsigned). All order hints are syntax elements previously read from the frame headers in the bitstream (not a derived value). Note : for tiles of a KEY_FRAME, saved order hints should be all set to a value of 0.
6	31:24	**Reserved** Access: RO Format: MBZ
	23:16	**Saved Order Hints for All References[2][6]** Format: U8
	15:8	**Saved Order Hints for All References[2][5]** Format: U8
	7:0	**Saved Order Hints for All References[2][4]** Format: U8
7	31:24	**Saved Order Hints for All References[3][3]** Format: U8
	23:16	**Saved Order Hints for All References[3][2]** Format: U8
	15:8	**Saved Order Hints for All References[3][1]** Format: U8
	7:0	**Saved Order Hints for All References[3][0]** Format: U8 This array specifies the saved order hints for all references for the purpose of computing the Motion Field Projections. Array indices are defined as [a reference frame index][the 7 reference indices of the reference frame]. Hence, the array indices' range are [0..6] [0..6]. Index value = 0, is set for LAST frame. Index value = 6 is set for ALTREF frame. Since a current frame can have up to 7 references, when each reference was previously decoded as a current frame, it could also have up to 7 references (references of a reference frame), hence totally max. there can be 7x7=49 references in the past. With each reference has its own order hint (previously read from the bitstream), a total of max. 49 order hints need to be saved and programmed here. Order Hint is acting like a form of time step, and can take on a value of [0..255] (8-bits unsigned). All order hints are syntax elements previously read from the frame headers in the bitstream (not a derived value).

AVP_INTER_PRED_STATE

		Note : for tiles of a KEY_FRAME, saved order hints should be all set to a value of 0.
8	31:24	**Reserved** Access: RO Format: MBZ
	23:16	**Saved Order Hints for All References[3][6]** Format: U8
	15:8	**Saved Order Hints for All References[3][5]** Format: U8
	7:0	**Saved Order Hints for All References[3][4]** Format: U8
9	31:24	**Saved Order Hints for All References[4][3]** Format: U8
	23:16	**Saved Order Hints for All References[4][2]** Format: U8
	15:8	**Saved Order Hints for All References[4][1]** Format: U8
	7:0	**Saved Order Hints for All References[4][0]** Format: U8 This array specifies the saved order hints for all references for the purpose of computing the Motion Field Projections. Array indices are defined as [a reference frame index][the 7 reference indices of the reference frame]. Hence, the array indices' range are [0..6] [0..6]. Index value = 0, is set for LAST frame. Index value = 6 is set for ALTREF frame. Since a current frame can have up to 7 references, when each reference was previously decoded as a current frame, it could also have up to 7 references (references of a reference frame), hence totally max. there can be 7x7=49 references in the past. With each reference has its own order hint (previously read from the bitstream), a total of max. 49 order hints need to be saved and programmed here. Order Hint is acting like a form of time step, and can take on a value of [0..255] (5-bits unsigned). All order hints are syntax elements previously read from the frame headers in the bitstream (not a derived value). Note : for tiles of a KEY_FRAME, saved order hints should be all set to a value of 0.
10	31:24	**Reserved** Access: RO Format: MBZ
	23:16	**Saved Order Hints for All References[4][6]** Format: U8
	15:8	**Saved Order Hints for All References[4][5]** Format: U8

AVP_INTER_PRED_STATE

	7:0	**Saved Order Hints for All References[4][4]** Format: U8
11	31:24	**Saved Order Hints for All References[5][3]** Format: U8
	23:16	**Saved Order Hints for All References[5][2]** Format: U8
	15:8	**Saved Order Hints for All References[5][1]** Format: U8
	7:0	**Saved Order Hints for All References[5][0]** Format: U8 This array specifies the saved order hints for all references for the purpose of computing the Motion Field Projections. Array indices are defined as [a reference frame index][the 7 reference indices of the reference frame]. Hence, the array indices' range are [0..6] [0..6]. Index value = 0, is set for LAST frame. Index value = 6 is set for ALTREF frame. Since a current frame can have up to 7 references, when each reference was previously decoded as a current frame, it could also have up to 7 references (references of a reference frame), hence totally max. there can be 7x7=49 references in the past. With each reference has its own order hint (previously read from the bitstream), a total of max. 49 order hints need to be saved and programmed here. Order Hint is acting like a form of time step, and can take on a value of [0..255] (8-bits unsigned). All order hints are syntax elements previously read from the frame headers in the bitstream (not a derived value). Note : for tiles of a KEY_FRAME, saved order hints should be all set to a value of 0.
12	31:24	**Reserved** Access: RO Format: MBZ
	23:16	**Saved Order Hints for All References[5][6]** Format: U8
	15:8	**Saved Order Hints for All References[5][5]** Format: U8
	7:0	**Saved Order Hints for All References[5][4]** Format: U8
13	31:24	**Saved Order Hints for All References[6][3]** Format: U8
	23:16	**Saved Order Hints for All References[6][2]** Format: U8
	15:8	**Saved Order Hints for All References[6][1]** Format: U8

AVP_INTER_PRED_STATE		
	7:0	**Saved Order Hints for All References[6][0]** Format: U8 This array specifies the saved order hints for all references for the purpose of computing the Motion Field Projections. Array indices are defined as [a reference frame index][the 7 reference indices of the reference frame]. Hence, the array indices' range are [0..6] [0..6]. Index value = 0, is set for LAST frame. Index value = 6 is set for ALTREF frame. Since a current frame can have up to 7 references, when each reference was previously decoded as a current frame, it could also have up to 7 references (references of a reference frame), hence totally max. there can be 7x7=49 references in the past. With each reference has its own order hint (previously read from the bitstream), a total of max. 49 order hints need to be saved and programmed here. Order Hint is acting like a form of time step, and can take on a value of [0..255] (8-bits unsigned). All order hints are syntax elements previously read from the frame headers in the bitstream (not a derived value). Note : for tiles of a KEY_FRAME, saved order hints should be all set to a value of 0.
14	31:24	**Reserved** Access: RO Format: MBZ
	23:16	**Saved Order Hints for All References[6][6]** Format: U8
	15:8	**Saved Order Hints for All References[6][5]** Format: U8
	7:0	**Saved Order Hints for All References[6][4]** Format: U8

AVP_PAK_INSERT_OBJECT

AVP_PAK_INSERT_OBJECT	
Source:	VideoCS
Length Bias:	2

It is an encoder only command, operating at bitstream level, before and after SliceData compressed bitstream. It is setup by the header and tail present flags in the Slice State command. If these flags are set and no subsequent PAK_INSERT_OBJECT commands are issued, the pipeline will hang.
The AVP_ PAK_ INSERT _OBJECT command supports both inline and indirect data payload, but only one can be active at any time. It is issued to insert a chunk of bits (payload) into the current compressed bitstream output buffer (specified in the AVP_PAK-BSE Object Base Address field of the AVP_IND_OBJ_BASE_ADDR_STATE command)starting at its current write pointer bit position. Hardware will keep track of this write pointer's byte position and the associated next bit insertion position index.
It is a variable length command when the payload (data to be inserted) is presented as inline data within the command itself. The inline payload is a multiple of 32-bit (1 DW), as the data bus to the compressed bitstream output buffer is 32-bit wide.
The payload data is required to be byte aligned on the left (first transmitted bit order) and may or may not be byte aligned on the right (last transmitted bits). The command will specify the bit offset of the last valid DW. Note that : Stitch Command is used if the beginning position of data is in bit position. When PAK Insert Command is used the beginning position must be in byte position.
Multiple insertion commands can be issued back to back in a series. It is host software's responsibility to make sure their corresponding data will properly stitch together to form a valid bitstream.
Internally, AVP hardware will keep track of the very last two bytes' (the very last byte can be a partial byte)values of the previous insertion. It is required that the next Insertion Object Command or the next PAK Object Command to perform the start code emulation sequence check and prevention 0x03 byte insertion with this end condition of the previous insertion.
The payload data may have already been processed for start code emulation byte insertion, except the possibility of the last 2 bytes plus the very last partial byte (if any). Hence, when hardware performing the concatenation of multiple consecutive insertion commands, or concatenation of an insertion command and a PAK object command, it must check and perform the necessary start code emulation byte insert at the junction.
Data to be inserted can be a valid NAL units or a partial NAL unit. It can be any encoded syntax elements bitdata before the encoded Slice Data (PAK Object Command) of the current Slice - SPS NAL, PPS NAL, SEI NAL and Other Non-Slice NAL, Leading_Zero_8_bits (as many bytes as there is), Start Code , Slice Header. Any encoded syntax elements bit data after the encoded Slice Data (PAK Object Command) of the current Slice and prior to the next encoded Slice Data of the next Slice or prior to the end of the bitstream, whichever comes first Cabac_Zero_Word or Trailing_Zero_8bits (as many bytes as there is).
Certain NAL unit has a minimum byte size requirement. As such the hardware will optionally (enabled by SLICE STATE Command) determines the number of CABAC_ZERO_WORD to be inserted to the end of the current NAL, based on the minimum byte size of a NAL and the actual bin count of the encoded Slice. Since prior to the CABAC_ZERO_WORD insertion, the RBSP or EBSP is already byte-aligned, so each CABAC_ZERO_WORD insertion is actually a 3-byte sequence 0x00 00 03.
Context switch interrupt is not supported by this command.

DWord	Bit	Description

AVP_PAK_INSERT_OBJECT

0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 3h Codec/Engine Name Format: OpCode Codec/Engine Name = AVP = 3h
	22:16	**Media Instruction Command** Default Value: 22h AVP_PAK_INSERT_OBJECT Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Default Value: [1h, FFFh] DWORD_COUNT_n Format: =n (Excludes Dwords 0, 1) =Total Length - 2, soDWord Length = X, where X is in the size of the payload in DWs 2..n which has the range of [1,4095]
1	31	**Indirect Payload Enable** Format: Enable Payload(header) must be inline only so this bit set to MBZ. **Value** | **Name** | **Description** 0 | inline payload is used | 1 | indirect payload is used | Indirect payload is not supported so this value must be zero
	30:18	**Reserved** Access: RO Format: MBZ
	17:16	**DataByteOffset - SrcDataStartingByteOffset[1:0]** Format: U2 Source Data Starting Byte Position within the very first inline DW.
	15:14	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">AVP_PAK_INSERT_OBJECT</th></tr>
<tr><td></td><td>13:8</td><td><b>DataBitsInLastDW - SrCDataEndingBitInclusion[5:0]</b>
<table><tr><td>Format:</td><td>U6</td></tr></table>
Source Data to be included in the very last inline DW. Follows the MSBit is the upper bit of each byte within the DW. The lower byte is actually processed first. For example, SrCDataEndingBitInclusion = 9, bit 7:0 and bit 15 are included as valid header data.
The Driver has to give byte aligned Data for the last inline DW. (the driver pads Zeros to next byte boundary to the original header if it was not byte aligned on the last inline DW).
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[1,32]</td><td></td></tr></table></td></tr>
<tr><td></td><td>7:3</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>2</td><td><b>LastHeaderFlag - LastSrcHeaderDataInsertCommandFlag</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
To process a series of consecutive insertion commands, this flag (=1) indicates the current command is the last 'header' insertion in the series.
If a tile/tile group has multiple headers, then set this flag “LastHeaderFlag – LastSrcHeaderDataInsertCommandFlag” to 1 on the last header.
Assumed all the headers are byte aligned.</td></tr>
<tr><td></td><td>1</td><td><b>EndOfHeaderInsertionFlag - LastDstDataInsertCommandFlag</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
No more insertion command and no more PAK-OBJECT command follows. Flush data out to memory.</td></tr>
<tr><td></td><td>0</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>2..n</td><td>127:0</td><td><b>Indirect Payload</b>
<table><tr><td>Exists If:</td><td>([Indirect Payload Enable]==1)</td></tr><tr><td>Format:</td><td>AVP_PAK_INSERT_OBJECT_INDIRECT_PAYLOAD</td></tr></table></td></tr>
<tr><td></td><td>31:0</td><td><b>Inline PayLoad</b>
<table><tr><td>Exists If:</td><td>([Indirect Payload Enable]==0)</td></tr><tr><td>Format:</td><td>U32</td></tr></table>
Actual Data (inline) to be inserted to the output bitstream buffer.</td></tr>
</table>

AVP_PAK_OBJECT

AVP_PAK_OBJECT

Source:	VideoCS
Length Bias:	1

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: Opcode
	28:27	**Pipeline Type** Default Value: 2h Format: Opcode
	26:23	**Media Instruction Opcode** Default Value: 3h Codec/Engine Name Format: OpCode Codec/Engine Name = AV1 = 3h
	22:16	**Media Instruction Command** Default Value: 21h AVP_PAK_OBJECT Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 3h Name:
1	31:16	**Current SB Y Addr** Format: U16 Description Supports 16kx16k frame size so valid bits are 7:0 and the upper bits are for future use.
	15:0	**Current SB X Addr** Format: U16 Description Supports 16kx16k frame size so valid bits are 7:0 and the upper bits are for future use.

<table>
<tr><th colspan="3">AVP_PAK_OBJECT</th></tr>
<tr><td>2</td><td>31</td><td><b>LastSBofTile</b>
Indicates if this SB is last of a Tile</td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>29:24</td><td><b>CU count minus1</b><table><tr><td>Format:</td><td>U6</td></tr></table>Number of CUs in the current SB = CU_count_minus1 + 1. Minimum, there must be 1 CU in a SB.</td></tr>
<tr><td></td><td>23:17</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>16</td><td><b>SBForceZeroCoeff/Time Budget Overflow Occurred</b><table><tr><td>Format:</td><td>Enable</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

AVP_PIC_STATE

AVP_PIC_STATE

Source:	VideoCS
Length Bias:	2

All AVP_PIC_STATE should stay the same for the whole frame even if AVP_PIC_STATE is re-programmed for every tiles.
The bitfields of AVP_PIC_STATE are defined either from
1) syntax elements of the uncompressed sequence header (received from sequence_header_obu) and of the uncompressed frame header (received from frame_header_obu),
2) or, parameters derived from 1).
Note : Bitstreams may contain several copies of the frame header (there can only be one frame_header_obu, but multiple redundant_frame_header_obu)interspersed with tile_group_obu to allow for greater error resilience. However, the copies must contain identical contents to the original frame_header_obu.
Note : there should be only one sequence_header_obu per video sequence.
Note : AVP pipeline is invoked to decode a frame from the bitstream, only if that frame has show_existing_frame flag (syntax element in the frame header) set to 0. For the case that show_existing_frame flag is set to 1, application and driver process the frame instead, no block level decoding is needed.
Note : Unlike VP9, AV1 does not have a compressed header. All the syntax elements defined in the AV1 sequence and frame level headers are not arithmetic coded, hence application and driver can directly read them off from the bitstream.
Note : the values of the sequence header/level syntax elements and their derived parameters are to last throughout all frames in the video sequence, until the next Sequence Header OBU is received that may change them. But some sequence header/level syntax elements or their derived parameters may further qualified by frame header/level syntax elements and their derived parameters, then these type of syntax elements and their derived parameters can be changed frame to frame.
Note : the values of the frame header/level syntax elements and their derived parameters can be changed from frame to frame.
Note : there are some syntax elements and their derived parameters can be changed only at KEY FRAME. Hence, the values of these type of syntax elements and their derived parameters can last for the entire GOP, i.e. until the next KEY FRAME that may change them.
Note : there is no separate profile for Still Picture. Still Picture is coded and decoded as a KEY FRAME, with all coding tools supported (tiling, all post in-loop filters, film grain injection, monochrome, intraBC, palette prediction mode, etc.). There is no restriction in coding Still Picture as a KEY FRAME.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 3h Codec/Engine Name Format: OpCode

<table>
<tr><th colspan="3">AVP_PIC_STATE</th></tr>
<tr><td></td><td></td><td>Codec/Engine Name = AVP = 3h</td></tr>
<tr><td></td><td>22:16</td><td><b>Media Instruction Command</b>
<table>
<tr><td>Default Value:</td><td>30h AVP_PIC_STATE</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>11:0</td><td><b>Dword Length</b>
<table>
<tr><td>Format:</td><td>=n</td></tr>
</table>
(Excludes Dwords 0, 1).
<table>
<tr><th>Value</th><th>Name</th><th>Programming Notes</th></tr>
<tr><td>31h</td><td>Decoder DW Length</td><td>Only Up to DW12 should be programmed for decoder</td></tr>
<tr><td>48h</td><td>Encoder DW Length</td><td>All DWs should be programmed for encoder</td></tr>
</table></td></tr>
<tr><td>1</td><td>31:16</td><td><b>Frame Height In Pixel Minus 1</b>
<table>
<tr><td>Format:</td><td>U16</td></tr>
</table>
Specifies the height of the frame to be decoded in unit of pixel. It is the same as the frame height in luma samples.
AV1 supports up to 64Kx64K frame size. Intel supports up to 16Kx16K frame size. Valid range [15, 16383].
Min frame height is 16pixels. Hence, Luma is 16and Chroma is 8(in 4:2:0).
It is a frame-level derived parameters, after taking into account of the syntax elementmax_frame_height in the sequence header and frame_size_override_flag in the frame header. For SWITCH Frame,frame_size_override_flag is always set to 1; for all other frame types (KEY Frame, INTRA-ONLY Frame and INTER Frame),frame_size_override_flag is read from the bitstream (and can be 0 or 1).For inter-frame, it also takes into account of the possibility of using frame size inferred from a reference frame.
Note : this field is not affected by Horizontal Super-Resolution coding.
Note : this frame height may be odd or even number, and may not be divisible by 4, 8 , superblock size, tile size, or LRU size.
Note : if frame height is an odd number, the corresponding height of the Chroma planes in 4:2:0 is rounded up. For example, if Luma height is 13 pixels, Chroma height is 7 pixels.
Note : internally the frame height is rounded up to be divisible by 8 in both the HW encoder and decoder. The padding in the encoder side is not normative, but will be coded into the bitstream. When decoder reconstructs the decoded frame from the bitstream, the padding is being reconstructed as well. The padding at the right and bottom borders of the decoded frame can be cropped away before display. But they are not cropped away when the decoded frame is added into the DPB as a reference frame.
Note : this is NOT the Render Frame Height, which is used to crop the decoded frame size for display.</td></tr>
</table>

AVP_PIC_STATE

	15:0	**Frame Width In Pixel Minus 1** Format: U16 Specifies the width of the decoded frame in unit of pixel. It is the same as the frame width in uma samples. AV1 supports up to 64Kx64K frame size. Intel supports up to 16Kx16K frame size. Valid range [15, 16383]. Min frame width is 16pixels. Hence, Luma is 16and Chroma is 8(in 4:2:0). It is a frame-level derived parameters, after taking into account of the syntax elementmax_frame_width in the sequence header and frame_size_override_flag in the frame header. For inter-frame and s-frame, it also takes into account of the possibility of using frame size inferred from a reference frame. When the Horizontal Super-Resolution is active, this frame width is the downscaled frame width. Note : if Horizontal-Only Super-Resolution is ON, this field specifies the reduced width of the downscaled frame size. The number of Superblocks to be processed per row in this case is derived from this reduced frame width. The tile configuration is also specified in this reduced frame width. But after deblocker and CDEFfiltering (if either or both are active), this reduced frame width is scaled back up to the original frame width for the subsequent Loop Restoration filtering (if active), which then is outputted for display or become a reference frame. Hence, all reference frames in the DPB are always stored in full frame resolution, and dynamic on-the-fly frame resizing is always invoked during Motion Comp at block level, which is coded with the reduced frame width. Note that the upscaled frame width cannot exist 16K range. upscaling frame width = reduced frame width * upscaling factor. The upscaled frame width is provided in the AVP_INLOOP_FILTER_STATE Command. Note that this frame width may be odd or even number, and may not be divisible by 4, 8 , superblock size, tile size, or LRU size. Note : if frame width is an odd number, the corresponding width of the Chroma planes in 4:2:0 is rounded up. For example, if Luma width is 13 pixels, Chroma width is 7 pixels. Note : internally the frame width is rounded up to be divisible by 8 in both the HW encoder and decoder. The padding in the encoder side is not normative, but will be coded into the bitstream. When decoder reconstructs the decoded frame from the bitstream, the padding is being reconstructed as well. The padding at the right and bottom borders of the decoded frame can be cropped away before display. But they are not cropped away when the decoded frame is added into the DPB as a reference frame. Note : this is NOT the Render Frame Width, which is used to crop the decoded frame size for display.
2	31:25	**Reserved** Access: RO Format: MBZ

AVP_PIC_STATE

<table>
<tr><td></td><td>24</td><td><b>Enable Bistream Stitching in hardware</b>
If set to 1, hardware will output encoded bitstream starting from end of last cacheline of previous tile
There is no context switching allowed between TILEs when this bit is set to 1
This bit should not be set to 1 in scalability mode
If set to 1, MinFrameSize[15:0] > 512bits + Header Size (Note: MinFrameSize is part of the this command @dword63, bits[15:0]
Encoder Only</td></tr>
<tr><td></td><td>23</td><td><b>Header Present Flag</b>
This bit indicates frave level header present before SB level stream
Encoder Only</td></tr>
<tr><td></td><td>22</td><td><b>Tail Present Flag</b>
This bit indicates Tail Present at the end of Frame
Encoder Only</td></tr>
<tr><td></td><td>21</td><td><b>Sequence Enable Joint Compound Flag</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
It specifies whether the Joint Compound coding tool is enabled for the video sequence, or not.
It is the sequence level syntax element, enable_jnt_comp, which is only present in the bitstream when the sequence level syntax element enable_order_hint is set to 1. It is defaulted to0, whenenable_order_hint is set to 0.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td><b>[Default]</b></td><td>Indicates that the distance weights process is NOT used for inter prediction.</td></tr>
<tr><td>1h</td><td></td><td>Indicate that the distance weights process may be used for inter prediction.</td></tr>
</table></td></tr>
<tr><td></td><td>20</td><td><b>Sequence Enable Masked Compound Flag</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
It specifies whether the Masked Compound coding tool is enabled for the video sequence, or not.
It is the sequence level syntax element, enable_masked_compound.
Valid only in Decoder Mode
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td><b>[Default]</b></td><td>Indicates that the mode info for inter blocks do NOT contain the block level syntax element compound_type and others.</td></tr>
<tr><td>1h</td><td></td><td>Indicates the block level syntax elements: compound_type and others, are present in the bitstream.</td></tr>
</table></td></tr>
<tr><td></td><td>19</td><td><b>Sequence Enable Inter-Intra Compound Flag</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
It specifies whether the Inter-Intra Compound coding tool is enabled for the video sequence, or not.
It is the sequence level syntax element, enable_interintra_compound.</td></tr>
</table>

<table>
<tr><th colspan="3">AVP_PIC_STATE</th></tr>
<tr><td></td><td></td><td>Valid in Decoder Mode only
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>[Default]</td><td>Indicates that the mode info for inter blocks do NOT contain the block level syntax element interintra and others.</td></tr>
<tr><td>1h</td><td></td><td>Indicates that the block level syntax elements: interintra and others, are present in the bitstream.</td></tr>
</table></td></tr>
<tr><td></td><td>18</td><td><b>Sequence Enable Dual_Filter Flag</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
It specifies whether the Motion Comp horizontal and vertical interpolation filters are specified independently in the bitstream at the block level (inter-coded block)or not, for the video sequence.
Set to 0 ; indicate only ONE MC filter type is specified in the bitstream, which is then used in both directions . (Default)
Set to 1 ; indicate the MC filter type may be specified independently in the horizontal and vertical directions.
It is the sequence level syntax element, enable_dual_filter.
Note : MC filter type can be [EIGHTTAP, EIGHTTAP_SMOOTH, EIGHTTAP_SHARP, BILINEAR, and SWITCHABLE].
Must be set to 0 in encoder mode</td></tr>
<tr><td></td><td>17</td><td><b>Sequence Enable Intra Edge Filter Flag</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
It specifies whether the Intra Edge Filtering process is enabled for the video sequence or not.
Set to 0 ; indicate that the Intra Edge Filter tool is DISabled. (Default)
Set to 1 ; indicate that the Intra Edge Filter tool is ENabled.
It is the sequence level syntax element, enable_intra_edge_filter.</td></tr>
<tr><td></td><td>16</td><td><b>Sequence Enable Filter_Intra Flag</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
It specifies if the Filter_Intra coding tool is enabled for the video sequence.
Set to 0 ; indicate the block level syntax element: use_filter_intra, is NOT present in the bitstream. (Default)
Set to 1 ; indicate the block level syntax element: use_filter_intra, is present in the bitstream.
It is the sequence level syntax element, enable_filter_intra.
Note : if both enable_filter_intra and use_filter_intra are set to 1, the corresponding block is coded with filter_intra
Must be set to 0 in encoder mode</td></tr>
<tr><td></td><td>15:13</td><td><b>Reserved (for the expansion of Sequence Order Hint Bits Minus1)</b>
<table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>12:10</td><td><b>Sequence Order Hint Bits Minus1</b>
<table><tr><td>Format:</td><td>U3</td></tr></table>
It specifies the number of bits to be read from the bitstream for the frame header syntax elements: order_hint and ref_order_hint[i=0 to 6].</td></tr>
</table>

AVP_PIC_STATE

		It is in the range of [0..7] . It is the sequence level syntax element, order_hint_bits_minus1, which is only present in the bitstream if the sequence level syntax element enable_order_hint_flag is set to 1. It is defaulted to 0, if not present. It can be used to generate the bitmask in computing the relative distance between two order hints. Note: it is also used for other purposes, as detail in the bitfield Sequence Enable Order Hint Flag.
	9	**Sequence Enable Order Hint Flag** Format: U1 Set to 1 ;enables the use of Order Hint in the decoding process of all INTER frames in the video sequence. Set to 0 ; disables the use of Order Hint. (Default) It is the sequence level syntax element, enable_order_hint. It controls 1) the bitstream reading of the syntax elements: enable_jnt_comp, enable_ref_frame_mvs, and order_hint_bits_minus1 in the sequence header. 2) the bitstream reading of the syntax elements: ref_order_hint[i=0 to 6] and frame_refs_short_signaling in the frame header. 3) the setting of reference frames in the frame header for the decoding of the current frame. Note : these 2 sequence level syntax elements: enable_order_hint flag and the 3-bits order_hint_bits_minus1 are used in : 1) skip_mode derivation, 2) compound prediction temporal weighing factor derivation, 3) motion field estimation process and MV projection, 4) motion_field MV storage setting, 5) reference frame bias derivation, 6) forward, second_forward, and backward reference frame selection, 7) and forward, second_forward and backward reference frame order hint setting.
	8:7	**Sequence Superblock Size Used** Format: U2 It specifies one of the two possible Superblock sizes that is used to code the video sequence. Set to 0, if the SuperBlock size is 64x64 pixels. (Default) Set to 1, if the SuperBlock size is 128x128 pixels. Value 2-3 are reserved. It is the sequence level syntax element, use_128x128_superblock; and is also named as sb_size. Supports SB size 64x64 only in encoder mode
	6	**Reserved** Access: RO Format: MBZ
	5	**Reserved (for expansion of Sequence Pixel Bit-Depth Idc)**

AVP_PIC_STATE

		Format: MBZ
	4:3	**Sequence Pixel Bit-Depth Idc** Format: U2 It specifies the pixel bit depth for all frames in the video sequence being decoded. [4:3] = 00 ; specifies 8-bit per pixel (bpp). (Default) [4:3] = 01 ; specifies 10-bit per pixel. [4:3] = 10 ; specifies 12-bit per pixel It is a sequence-level parameter derived from the sequence header syntax elements:seq_profile,high_bitdepth and twelve_bit. Note : Only Bit-Depth = 8 and 10 are allowed for this generation of AVP. Note : refer to the description under Sequence Chroma Sampling Format for detail on allowed bit depth for each profile.
	2	**Reserved (for expansion of Chroma SubSampling Format)** Format: MBZ
	1:0	**Sequence Chroma SubSampling Format** Format: U2 It specifies the chroma subsampling format for all frames in the video sequence being decoded. [1:0] = 00 ; stands for Monochrome 4:0:0, no Chroma planes at all, but [subsampling_x and subsampling_y] is defaulted to [1, 1], as only Profile 0 can support monochrome video coding. [1:0] = 01 ; stands for 4:2:0, with[subsampling_x and subsampling_y] defining as[1, 1]. It is supported in all profiles (seq_profile=0, 1, 2 - syntax element in the sequence header) [1:0] = 10 ; stands for 4:2:2, with[subsampling_x and subsampling_y] defining as[1, 0]. It is supported only in seq_profile=2. [1:0] = 11 ; stands for 4:4:4 with[subsampling_x and subsampling_y] defining as[0, 0]. It is supported in both seq_profile=1 and 2. It is a sequence-level parameter derived from the sequence header syntax elements: seq_profile, subsampling_x, subsampling_y, monochome, high_bitdepth and twelve_bit. Default is 1, i.e. 4:2:0.. Note : AV1 supports 3 profiles: seq_profile Bit_depth Chroma Subsampling 0 (Main Profile) 8 / 10 YUV 4:2:0 and 4:0:0 1 (High Profile) 8 / 10 YUV 4:4:4 (4:0:0 is not allowed) 2 (Pro Profile) 8 / 10 /12 YUV 4:2:2 AND 12 YUV 4:2:0/4:4:4/4:0:0 Note : for AV1 decoder: • A profile 0 compliant decoder must be able to decode all bitstreams labeled profile 0 • A profile 1 compliant decoder must be able to decode all bitstreams labeled profile 0 or 1

<table>
<tr><th colspan="3">AVP_PIC_STATE</th></tr>
<tr><td></td><td></td><td><table><tr><td><ul><li>A profile 2 compliant decoder must be able to decode all bitstreams labeled profile 0, 1, or 2</li></ul></td></tr><tr><td>"01" -- Chroma Sampling 4:2:0 is supported.</td></tr></table>
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>1h</td><td>4:2:0</td><td>Chroma Sampling 4:2:0</td></tr></table></td></tr>
<tr><td>3</td><td>31</td><td><b>Reserved (for future expansion of Primary Reference Frame Idx)</b><table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>30:28</td><td><b>Primary Reference Frame Idx</b><table><tr><td>Format:</td><td>U3</td></tr></table>
It specifies which one of the 7 possible reference frames [reference frame ID0 to 6]contains the frame context(CDF table set for all syntax elements) and other state that should be loaded at the start of the frame.
The normal range of values for Primary Reference Frame is [0 to 6]. The value of 7 (defined as PRIMARY_REF_NONE) is used to signal when there is no primary reference frame.
It is the frame level syntax element, primary_ref_frame, which is present in the bitstream when (! FrameIsIntra && ! error_resilient_mode). If it is not present in the bitstream, it is defaulted to PRIMARY_REF_NONE (=7).
There is no primary reference frame:
1) if (FrameIsIntra || error_resilient_mode). Hence only INTER Frame and SWITCH Frame can have primary reference frame ID specified in the bitstream when not in error resilient mode.
2) Or when it receives a value of 7from the bitstream.
Note : load_cdfs(ref_frame_idx[primary_ref_frame])is a function call that indicates that the frame context (CDF table set)is loaded from the frame context attached to one of the 8 possible reference frames in the DPB.
Note : load_previous( ) is a function call that indicates that information from a previous frame may be loaded for use in decoding the current frame.
Note : if (primary_ref_frame == PRIMARY_REF_NONE), it is required to set segmentation_update_map = 1 , segmentation_temporal_update = 0 , and segmentation_update_data = 1.
Note : If primary_ref_frame is set to PRIMARY_REF_NONE, it is a requirement of bitstream conformance that loop_filter_delta_update is equal to 1.</td></tr>
<tr><td></td><td>27:24</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>23</td><td><b>Allow IntraBC Flag</b><table><tr><td>Format:</td><td>U1</td></tr></table>
It specifies whether intra block copy prediction mode is allowed to be used in the current frame or not.
Set to 0 to disallow intra block copy prediction mode. (Default)</td></tr>
</table>

<table>
<tr><th colspan="3">AVP_PIC_STATE</th></tr>
<tr><td></td><td></td><td>Set to 1 to allow intra block copy prediction mode.
It is the frame level syntax element, allow_intrabc. It is present in the bitstream, only if the sequence syntax element allow_screen_content_tools is set to 1 AND there is no super-resolution frame width scaling (but super-resolution can still be enabled). When it is not present in the bitstream, it is defaulted to 0.
Note : intra block copy is only allowed in KEY Frame and INTRA-ONLY Non-Key Frame. For all other frame types (INTER Frame and SWITCH Frame), this flag is set to 0.
Note : when this flag is set to 1, all post in loop filters (deblocker, CDEF, Super-Resolution and Loop Restoration) are all disabled.
Note : when this flag is set to 1, force_integer_mv is set to 1 in KEY Frame and INTRA-ONLY Non-Key Frame.
Valid only in Decoder Mode</td></tr>
<tr><td></td><td>22</td><td><b>Error Resilient Mode Flag</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
It specifies whether all syntax decoding of the current frame is independent of the previous frames, or not.
Set to 0 to disable error resilient mode
Set to 1 to enable error resilient mode (for independent syntax decoding)
It is the frame-level syntax element, error_resilient_mode. Default is 0.
It is read from the bitstream for all frame types (KEY Frame, INTRA-ONLY Frame and INTER Frame), except when frame_type is set to SWITCH_FRAME, in which it is forced to 1 instead of reading from the bitstream.
When error resilient mode is set to 1 (active), Refresh Frame Context is set to0.
When error resilient is set to 0, Refresh Frame Context is read from the bit stream.
Valid only in Decoder Mode
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Disable</td></tr><tr><td>1</td><td>Enable</td></tr></table></td></tr>
<tr><td></td><td>21:20</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>19</td><td><b>IntraOnly Flag</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
It specifies if the current frame being decoded is a KEY Frame or a INTRA-ONLY Non-Key Frame.
It is a derived parameter from the frame level syntax element, frame_type, and is also named as FrameIsIntra.
IntraOnly Flag = (frame_type == INTRA_ONLY_FRAME) || (frame_type == KEY_FRAME).</td></tr>
<tr><td></td><td>18</td><td><b>Reserved (for the expansion of Frame Type)</b>
<table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>17:16</td><td><b>Frame Type</b></td></tr>
</table>

AVP_PIC_STATE		
		Format: U2 Specifies one of the four possible AV1 frame types. [17:16] = [00] ; specifies KEY Frame. (Default) [17:16] = [01] ; specifies INTER Non-Key Frame. [17:16] = [10] ; specifies INTRA-ONLY Non-Key Frame. [17:16] = [11] ; specifies SWITCH Non-Key Frame (or called S-Frame, is a different type of INTER Frame). It is the frame level syntax element, frame_type. Note: Encoder does not support INTRA-ONLYand S-Frame
	15	**Post Wiener Filtered Recon Pixels WriteoutEn** Set to 1, enable Post Winer Filtered Reconstructed Pixels write out to Memory for Motion Estimation purpose Set to 0, disable Valid in Encoder Mode
	14	**Post CDEF Filtered Recon Pixels Writeout En**
	13:9	**Reserved** Access: RO Format: MBZ
	8	**Large Scale Tile Enable Flag** Format: U1 Specify if the current tile decoding mode is the Large Scale Tile for VR application. It is set to 0 - for regular video decoding mode (Default) It is set to 1 - for Large Scale Tile decoding mode. Large Scale Tile is also known as ext-tile decoding mode. This field is a derived frame and tile level parameter. All tiles in a tile list are decoded in Large Scale Tile decoding mode. Valid in Decoder Mode Only
	7	**Reserved** Access: RO Format: MBZ
	6	**Frame Level Loop Restoration Filter Enable Flag** Format: U1 **Description** It specifies whether the Post In-Loop Loop Restoration Filter tool is enabled for the current frame, or not. Set to 0 ; indicate that the Loop Restoration Filter is DISabled for the current frame.(Default) Set to 1 ; indicate that the Loop Restoration Filter is ENabled for the current frame. Frame Level Loop Restoration Filter Enable Flag is an intel derived parameter, and is set to 1 when Luma frame_restoration_type != RESTORE_NONE || Chroma Cb frame_restoration_type != RESTORE_NONE || Chroma Cr frame_restoration_type !=

AVP_PIC_STATE

		RESTORE_NONE; It is derived from 1) the sequence level syntax element, enable_restoration, 2) the frame level derived coding parameters : frame level all_lossless and allow_intrabc, 3) and also the frame level syntax for loop restoration filter type of each color component. Although Use Loop Restoration Filter Flag = ! (AllLossless || allow_intrabc || !enable_restoration) can be used to signal the disabling of the loop restoration filter for the current frame, it is still possible at the frame level to read in the loop restoration filter type syntax elements that indicate the loop restoration filter is not enabled. To simplify hardware, this field is used to give HW the derived final decision if the loop restoration filter is enabled or not. When Large Scale Tile Enable flag is set to 1, this Loop Restoration Filterflag must set to 0, to disable the loop restoration filtering completely. Note : When individual frames in a video sequence are coded in Alllossless or with intraBC enabled, all post in-loop filtering processes (deblocker, CDEF, horizontal super-res, and loop restoration) should be disabled. Note : When Loop Restoration filter is disabled, all its filter types should be programmed to RESTORE_NONE. In Encoder Mode: This flag must be set to zero (No Loop Restoration Filter support)
	5	**Use Super-Res Flag** Format: U1 **Description** It specifies whether the Post In-Loop Horizontal-Only Super-Resolution tool is enabled for the current frame being decoded, or not. Set to 0 ; indicate that the Super-Res Frame Upscaling process is NOT performed. This sequence level setting cannot be overridden. (Default) Set to 1 ; indicate that the Super-Res Frame Upscaling process is to be performed. But this sequence level setting can be overridden when in frame coded lossless or when intraBC is enabled. It is derived from the sequence level syntax element, enable_superres, from the frame level syntax element of the same name: use_superres, and from the frame level derived coding parameters : frame coded_lossless and allow_intrabc. If the frame level syntax element of the same name, use_superres is not present in the bitstream, it is defaulted to 0. When Large Scale Tile Enable flag is set to 1, this Use Super-Res flag must set to 0, to disable super-resolution coding method. Note : it is legal to set the sequence level syntax element: enable_superresequal to 1, even when use_superres is set to 0 (i.e. no superres is performed)on all frames in the coded video sequence. Note : When individual frames in a video sequence are coded in coded_lossless or with intraBC enabled, all post in-loop filtering processes (deblocker, CDEF, horizontal super-res, and loop restoration) should be disabled. Note : When Super-res is disabled, the Denom of the scaling factor should be set

AVP_PIC_STATE

		to 8 (to have the same value as the default NUMERATOR). In Encoder Mode: This flag must be set to zero(no super-res).
	4	**Use CDEF Filter Flag** Format: U1 It specifies whether the Post In-Loop CDEF Filter tool is enabled for the current frame being decoded, or not. Set to 0 ; indicate that the CDEF Filter is DISabled. This sequence level setting cannot be overriden. (Default) Set to 1 ; indicate that the CDEF Filter is ENabled. But this sequence level setting can be overridden when in frame-level coded lossless or when intraBC is enabled. It is derived from the sequence level syntax element of the same name, enable_cdef and the frame level derived coding parameters : coded_lossless and allow_intrabc. Use CDEFFilter Flag = ! (Codedlossless|| allow_intrabc || !enable_cdef). When Large Scale Tile Enable flag is set to 1, this Use CDEF Filter flag must set to 0, to disable CDEF Filtering operation completely. Note : it is legal to set enable_cdef equal to 1 even when cdef filtering is not used on any frame in the coded video sequence. Note : When individualframes in a video sequence are coded in lossless or with intraBC enabled, all post in-loop filtering processes (deblocker, CDEF, horizontal super-res, and loop restoration) should bedisabled. Note : When CDEF filter is disabled, all its filter parameters should be reset to 0.
	3	**Reserved** Access: RO Format: MBZ
	2	**Allow Warped Motion Flag** Format: U1 Set to 0 ; indicates that the block level syntax element: motion_mode is NOT present in the bitstream. Hence, the block cannot signal for LOCALWARP coding. Set to 1 ; indicates that the block level syntax element: motion_mode is present in the bitstream. It is the frame level syntax element, allow_warp_motion, which is present in the bitstream only if thiscondition is met: (! FrameIsIntra && ! error_resilient_mode && enable_warped_motion). If it is not present in the bitstream, it is defaulted to 0. Note : motion_mode can take on a value of [SIMPLE. OBMC, or LOCALWARP]. LOCALWARP not supported in encoder mode
	1	**Force Integer MV Flag** Format: U1 Set to 0 ; indicates that motion vectors used in the Motion Comp can contain fractional bits (sub-pel precision). (Default) Set to 1 ; indicates that motion vectors used in the Motion Comp is always integers (NO sub-pel precision for Luma only). It is derived from thesequence level syntax elements: seq_choose_screen_content_tools andseq_force_screen_content_tools, and from the

AVP_PIC_STATE

		frame level syntax element : allow_screen_content_tools. if (allow_screen_content_tools = =1) AND (seq_force_integer_mv = = 2), read the value of force_integer_mv flag from the bitstream (in the frame header). If(allow_screen_content_tools = =1) AND (seq_force_integer_mv != 2) , force_integer_mv flag is set to the value of seq_force_integer_mv (which is read from the bitstream in the sequence header). if(allow_screen_content_tools = =0) , force_integer_mv is set to0. Note : for 4:2:0 and 4:2:2, the chroma subsampling factors (subsampling_x and subsampling_y) are also applied to the Luma MVs. Hence, integer Luma MVs will still give half-pel precision Chroma MVs (i.e. with fractional bit). Note : if intraBC is enabled, force_integer_mv is always set to 1 in the intra frames (KEY Frame or Intra-Only Non-Key Frame). Note : when force_integer_mv is set to 1, some fractional bits are still read for the translation components. However, these fractional bits will be discarded during the Setup Zero MV process. Valid only in decoder Mode
	0	**Allow Screen Content Tools Flag** Format: U1 Set to 0 ; indicates that the two block level screen content coding tools (palette prediction coding and intraBC coding). (Default) Set to 1 ; indicates that the block level syntax element: motion_mode is present in the bitstream. It is derived from the sequence level syntax elements: seq_choose_screen_content_tools andseq_force_screen_content_tools, and from the frame level syntax element of the same name: allow_screen_content_tools. ifseq_choose_screen_content_tools = = 1, read the value of allow_screen_ccontent_tools from the bitstream (in the frame header). Ifseq_choose_screen_content_tools = = 0, allow_screen_content_tools is set to the value of seq_force_screen_content_tools (which is read from the bitstream in the sequence header). Note : at the frame header, allow_screen_content_tools flag controls the setting of intraBC and integer_mv coding tools, but at the block level, it controls the use of the palette prediction coding mode. Valid in Decoder Mode only
4	31	**Reserved** Access: RO Format: MBZ
	30:24	**Y_dc_delta_q** Format: S6 **Programming Notes** 2's complement sign number of range -63 to +63.
	23:16	**Base Qindex**

AVP_PIC_STATE

		Format: U8 It can take on value range from 0 to 255. This is the same as the Y_ac_q, becasue y_ac_delta_q is always set to 0.
	15:14	**Reserved** Access: RO Format: MBZ

	13	**Segment ID Buffer Stream-Out Enable Flag**
		Format: U1
		It indicates if the processing of the current frame requires to write out the segment IDs at the block level to a segment ID Buffer.
		It is set to 1, if stream-out is going to happen.
		It is set to 0, if stream-out is not going to happen at all.
		It is an intel derived frame level parameters based on only frame level syntax and other derived parameters. Its meaning is defined as follows:
		if(SegmentIdStreamOutFlag)
		{
		HW needs to write out segment ID to Segment ID Write Buffer;
		}
		else
		{
		HW does not need to write out segment ID;
		}
		Driver is responsible to set this Segment ID Buffer Stream-Out Enable Flag based on the following conditions (using Driver variable names) :
		Frame Level derived parameter : (seg->enable = = 1) &&
		Frame Level SE : seg->update_map = = 1
		Valid in Decoder only Mode

	12	**Segment ID Buffer Stream-In Enable Flag**
		Format: U1
		It indicates if the processing of the current frame requires to read in the segment IDs at the block level from a segment ID Buffer.
		It is set to 1, if stream-in is going to happen.
		It is set to 0, if stream-in is not going to happen at all.
		It is an intel derived frame level parameters based on only frame level syntax and other derived parameters. Its meaning is defined as follows:
		Valid in Decoder only Mode
		if (SegmentMapIsZeroFlag)
		{
		Segment ID is all 0s.
		Driver wlill programSegmentIdStreamInFlagto 0. (HW checks on the SegmentMapIsZeroFlag=1, and set all temporal segment ID to 0)
		}
		else if (SegmentIdStreamInFlag)
		{

AVP_PIC_STATE

<table>
<tr><td></td><td></td><td>Segment ID is streamed in from Segment ID read buffer;
}
else
{
No Segment ID stream-in
}
Driver is responsible to set this Segment ID Buffer Stream-InEnable Flag based on the following conditions (using Driver variable names) :
Frame Level derived parameter : (seg->enable = = 1) &&
Frame Level SE : (primary_ref_frame != PRIMARY_REF_NONE) &&
Frame Level derived parameters :
((seg->update_map = = seg->temporal_update = = 1) ||
(seg->update_map = = seg->temporal_update = = 0)) &&
((DPB[current_frame].resolution = = DPB[primary_ref_frame].resolution) &&
(DPB[primary_ref_frame].seg_enable) &&
(DPB[primary_ref_frame].segment_ID_buffer != NULL))</td></tr>
<tr><td></td><td>11</td><td><b>Segment Map Is Zero Flag</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
<table>
<tr><th>Description</th></tr>
<tr><td>In a video sequence or within a GOP, some frames can have segmentation disabled. Their corresponding segment map is defaulted to have its content set to all 0. If later, this all zero segment map will be referenced by HW for temporal segment map prediction, it will be a waste of bandwidth to read in zeros.
Segment Map Is Zero flag is an intel added parameter at frame level.
If it is set to 1, it tells HW that the segment map is containing all zero. AVP HW will check this bit together with if segment map streamin is enabled, then HW will not actually read from the segment map buffer, but will internally generate the read of 0 instead, to save bandwidth.
Driver needs to keep track of which reference frame(s) (max 7) have segmentation disabled for decoding the current frame, and set this flag accordingly.
Driver is responsible to set this Segment Map Is Zero Flag based on the following conditions (using Driver variable names) :
Frame Level derived parameter : (seg->enable = = 1) &&
Frame Level SE : (primary_ref_frame != PRIMARY_REF_NONE) &&
Frame Level derived parameters :
((seg->update_map = = seg->temporal_update = = 1) ||
(seg->update_map = = seg->temporal_update = = 0)) &&
! [((DPB[current_frame].resolution = = DPB[primary_ref_frame].resolution) &&
(DPB[primary_ref_frame].seg_enable) &&
(DPB[primary_ref_frame].segment_ID_buffer != NULL))]</td></tr>
<tr><td>Decoder Only</td></tr>
</table></td></tr>
<tr><td></td><td>10</td><td><b>Frame Coded Lossless Mode</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
This bit Set to indicate lossless coding mode at frame level.</td></tr>
</table>

<table>
<tr><th colspan="3">AVP_PIC_STATE</th></tr>
<tr><td></td><td></td><td>Frame Coded Lossless Mode is set to 1, if all active segment's segment lossless flag are set to 1.
The equation for deriving coded lossless mode is presented in the AVP_SEGMENT_STATE Command.
AllLossless = CodedLossless && (FrameWidth == UpscaledWidth). The second condition in this equation is equivalent tohaving Super-res NOT enabled.
Only CodedLossless flag is sent to HW. AllLossless flag is not.
CodedLossless directly control the enabling/disabling of deblocker, CDEF in-loop filters.
But AllLossless is used to control the enabling/disabling of Loop Restoration filter. Hence, when super-res is ON, Loop Restoration filter can still be ON/OFF, regardless of CodedLossless.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Normal Mode</td></tr>
<tr><td>1</td><td>Coded Lossless Mode</td></tr>
</table></td></tr>
<tr><td></td><td>9:8</td><td><b>Delta Q RES</b>
<table><tr><td>Format:</td><td>U2</td></tr></table>
if delta_q_present_flag = 1, 2 bits from bitstream are read.
Delta_q_res = 0, 1, 2 and 3.
and the mulitple factor= 1 « (delta_q_res) = [1, 2, 4 or 8].
Here, only the 2 bits are programmed to the HW.</td></tr>
<tr><td></td><td>7</td><td><b>Delta Q Present Flag</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
if delta_q_present_flag = 1, delta_q_res is present in the bitstream - to specify SB level delta q.
In encoder mode, this flag must be set to 1 for BRC purpose so the QP can be changed at SB level.
However, it can be set to 0 for CQP across frame</td></tr>
<tr><td></td><td>6:4</td><td><b>Last Active Segment ID</b>
<table><tr><td>Format:</td><td>U3</td></tr></table>
It specifies the highest numbered segment id that has some enabled feature. This is used when decoding the segment id to only decode choices corresponding to used segments.</td></tr>
<tr><td></td><td>3</td><td><b>Pre-Skip Segment ID Flag</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
Set to 1 indicates that the segment id will be read before the skip syntax element.
SegIdPreSkip equal to 0 indicates that the skip syntax element will be read first.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This bit must be 1 in encoder mode</td></tr>
</table></td></tr>
<tr><td></td><td>2</td><td><b>Segmentation Temporal Update Flag</b>
<table><tr><td>Format:</td><td>U1</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">AVP_PIC_STATE</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>Indicates whether segID is decoding from bitstream or predicted from previous frame.</td></tr>
<tr><td>In encoder Mode it should use either from previous frame or streamIn</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0h</td><td>Decode segID from bitstream</td></tr>
<tr><td>1h</td><td>Get segID either from bitstream or from previous frame</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Decoder Only: For KEY_FRAME or INTRA_ONLY frame [OR ERROR_RESILIENCE], this bit should be set to "0".
[Temporary add in the condition ERROR_RESILIENCE, to be removed later]
Note: Driver should override this flag to "0" in KEY_FRAME or INTRA_ONLY frame even if this bit decoded from bitstream is different. This is for hardware optimization. This override does not affect bitstream decoding other than uncompressed header.</td></tr>
</table>
</td></tr>
<tr><td></td><td>1</td><td>
<b>Segmentation Update Map Flag</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
Set to 0 means that the segmentation map from the previous frame is used, and the current frame decode is not changing the segmentation map. (Default)
Set to 1 means that the segmentation map is updated during the decoding of the current frame. Hence SegmentIDs of current frame are streamout to a write only surface, which will be used as the segmentation map for a future frame(s).
It is the frame level syntax element, segmentation_update_map. It is present in the bitstream only if, the frame level syntax elements: segmentation is enabled AND primary_ref_frame is NOT PRIMARY_REF_NONE. If primary_ref_frame is set to PRIMARY_REF_NONE, segmentation_update_map is always set to 1.
Note : the segmentation map that has streamout (surface buffer) is attached to the current frame, after it has become a reference frame in the Display Buffer (DPB).
Note : HW needs to detect one of these 3 conditions (the current frame is in error resilient mode OR the current frame type is KEY Frame or INTRA-ONLY Frame), and if TRUE, then HW will not read the segment map for decoding the current frame, all segment ID is considered as 0. But during the decoding of the current frame, the segment map can still be modified is segmentation map update flag is ON.
</td></tr>
<tr><td></td><td>0</td><td>
<b>Segmentation Enable Flag</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
Indicate if segmentation is enabled or not
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0h</td><td>All blocks are implied to belong to segment 0</td></tr>
<tr><td>1h</td><td>SegID determination depends on segmentation_update_map setting</td></tr>
</table>
</td></tr>
<tr><td>5</td><td>31</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
</table>
</td></tr>
</table>

AVP_PIC_STATE

		Format: MBZ
	30:24	**V_ac_delta_q** Format: S6 **Programming Notes** 2's complement sign number of range -63 to +63.
	23	**Reserved** Access: RO Format: MBZ
	22:16	**V_dc_delta_q** Format: S6 **Programming Notes** 2's complement sign number of range -63 to +63.
	15	**Reserved** Access: RO Format: MBZ
	14:8	**U_ac_delta_q** Format: S6 **Programming Notes** 2's complement sign number of range -63 to +63.
	7	**Reserved** Access: RO Format: MBZ
	6:0	**U_dc_delta_q** Format: S6 **Programming Notes** 2's complement sign number of range -63 to +63.
6	31:28	**Reserved** Access: RO Format: MBZ
	27:24	**Reserved (for future expansion of Frame Order Hint)** Format: MBZ
	23:16	**Current Frame Order Hint** Format: U8

AVP_PIC_STATE

		It specifies "OrderHintBits" least significant bits of the expected output order for the current frame being decoded. It is the frame level syntax element, order_hint. Default value is 0. Order_hint is a variable bit length syntax element; its bit length is in the range of OrderHintBits=[1..8]. Hence, maximum order_hint can have a value of 255. Note: There is no requirement that OrderHint should reflect the true output order.
	15:8	**Reference Frame Sign Bias [i=0 to 7]** Format: U8 This is a bit array that specifies the Reference Frame Sign Bias for Reference 0 to 7 (only 1 to 7 are valid, OR 0 to 6 ???) Reference Picture 0 is INTRA frame and there is no sign bias for it. Bit 9 : Last Frame (Reference Picture 1) Bit 10: Last2 Frame (Reference Picture 2) Bit 11: Last3 Frame (Reference Picture 3) Bit 12: GoldenFrame (Reference Picture 4) Bit 13: Bwdref Frame (Reference Picture 5) Bit 14: Altref2 Frame (Reference Picture 6) Bit 15: Altref Frame (Reference Picture 7) It is a frame-level derived parameter for each reference frame, RefFrameSignBias[i=0 to 7]. 1-bit per reference frame. It is derived from the frame level parameters order hint and ref order hints. If the sequence syntax element enable_order_hint is set to 0, RefFrameSignBias[i=0 to 7] are all set to 0.
	7	**Use Reference Frame MV Set Flag** Format: U1 Set to 0, specifies that motion vector information from a previous frame (stored inside the temporal MV buffer) CANNOT be used when decoding the current frame. (Default) Set to 1, specifies that motion vector information from a previous frame can be usedwhen decoding the current frame. It is the frame level syntax element, use_ref_frame_mvs, which is present in the bitstream only if the sequence level syntax elements: enable_order_hint and enable_ref_frame_mvs are both set to 1 and the frame level syntax element: error_resilient_mode is set to 0. If it is not present in the bitstream, it is defaulted to 0.
	6	**Motion Mode Switchable Flag** Format: U1 Set to 0, specifies only SIMPLE motion mode can be used. (Default) Set to 1, specifies the motion mode being used is determined at the block level. It is the frame level syntax element, is_motion_mode_switchable. It is present only in INTER Frame or SWITCH Frame. For all other frame types (KEY Frame and INTRA-ONLY Frame), it is defaulted to 0. Note :Motion Mode for motion comp can be SIMPLE, OBMC or LOCALWARP.
	5	**Reserved (for future expansion of Mcomp Filter Type)**

AVP_PIC_STATE

<table>
<tr><td></td><td></td><td><table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>4:2</td><td><b>Mcomp Filter Type</b><table><tr><td>Format:</td><td>U3</td></tr></table>It specifies which Motion Compensation Filter type is to be used for the current frame.
It is a frame-level derived parameters. It is derived from the frame level syntax elements (is_filter_switchable flag and the 2-bits interpolation_filter).
Default is 0 (i.e. use the eight-tap basic filter).<table><tr><th>Value</th><th>Name</th></tr><tr><td>0h</td><td>Eight-tap</td></tr><tr><td>1h</td><td>Eight-tap-Smooth</td></tr><tr><td>2h</td><td>Eight-tap-Sharp</td></tr><tr><td>3h</td><td>Bilinear</td></tr><tr><td>4h</td><td>Switchable</td></tr></table></td></tr>
<tr><td></td><td>1</td><td><b>Frame Level Reference Mode Select</b><table><tr><td>Format:</td><td>U1</td></tr></table>Set to 0 specifies that all inter blocks in the current frame will use single prediction (SINGLE_REFERENCE). Default is 0, use SINGLE_REFERENCE.
Set to 1 specifies that the frame level prediction mode for inter blocks is REFERENCE_MODE_SELECT, which will cause reading the syntax element comp_mode at PartU level to specify whether to use single or compound reference prediction for that partU.
It is the frame level syntax element, reference_select. If it is NOT present in the bitstream, it is defaulted to 0.</td></tr>
<tr><td></td><td>0</td><td><b>Allow High Precision MV</b><table><tr><td>Format:</td><td>U1</td></tr></table>It specifies whether the high precision MV mode is used for the Luma Motion Vector prediction or not.
Set to 0, specifies that motion vectors are in quarter-pel precision. (Default, the normal mode)
Set to 1, specifies that motion vectors are in eighth-pel precision.
It is the frame-level syntax element, allow_high_precision_mv. It is present in the bitstream, only if the frame level parameter force_integer_mv is set to 0. If it is not present in the bitstream, it is default to 0.</td></tr>
<tr><td>7</td><td>31:24</td><td><b>Reference Frame Side [i=0 to 7]</b><table><tr><td>Format:</td><td>U8</td></tr></table>This is a bit array that specifies the Reference Frame Sidefor Reference 0 to 7.
Bit 24: Intra Frame (Reference Picture 0)
Bit 25: Last Frame (Reference Picture 1)
Bit 26: Last2 Frame (Reference Picture 2)
Bit 27: Last3 Frame (Reference Picture 3)
Bit 28: GoldenFrame (Reference Picture 4)</td></tr>
</table>

AVP_PIC_STATE

		Bit 29: Bwdref Frame (Reference Picture 5) Bit 30: Altref2 Frame (Reference Picture 6) Bit 31: Altref Frame (Reference Picture 7) It is a intel frame-level derived parameter for each reference frame, For each reference frame the corresponding bit is set to 0 when the corresponding bit in ref_frame_side is 0, else the bit will be set to 1. Each individual bit (i=0 to 7)of the original reference frame side parameter can take on a value of -1, 0, or 1. But intel version of the same parameter can only take on a value of 0 or 1. Both the original value of -1 and1 is mapped to 1 here, and the original value of 0 remains mapped to 0. Default all 8-bits are set to 0.
	23:13	**Reserved** Access: RO Format: MBZ
	12	**Reserved (for future expansion of Skip Mode Frame[1])** Format: MBZ
	11:9	**Skip Mode Frame [1]** Format: U3 It specifies the reference frames to use for compound prediction when skip_mode is set to 1. It is the frame level derived parameter SkipModeFrame[1]. Skip mode tries to use the closest forward (past) and backward (future) references (as measured by values in the RefOrderHint array). If no backward reference is found, then the second closest forward reference is used. If no forward reference is found, then skip mode is disabled.
	8	**Reserved (for future expansion of Skip Mode Frame[0])** Format: MBZ
	7:5	**Skip Mode Frame [0]** Format: U3 It specifies the reference frames to use for compound prediction when skip_mode is set to 1. It is the frame level derived parameter SkipModeFrame[0]. Skip mode tries to use the closest forward (past) and backward (future) references (as measured by values in the RefOrderHint array). If no backward reference is found, then the second closest forward reference is used. If no forward reference is found, then skip mode is disabled.
	4	**Skip Mode Present Flag** Format: U1 Set to 0 specifies that skip_mode will not be used for this frame. Default is 0, no PartU level skip flag. Set to 1 specifies that the syntax element skip_mode will be coded in the bitstream at the PartU level. It is the frame-level syntax element skip_mode_present. It is present in the bitstream

AVP_PIC_STATE

		based on an algorithm using RefOrderHint. If it is not present in the bitstream, it is defaulted to 0.
	3	**Reserved** Access: RO Format: MBZ
	2:1	**Frame Transform Mode** Format: U2 It specifies the Luma-only transform size to be used for the entire current frame to be decoded. 1) tx_mode=0 for ONLY_4X4, if the current frame is coded with frame coded lossless and only Hadamard 4x4 transform is being used for the whole frame. This is also applied to Chroma planes as well. 2) tx_mode=1 for TX_MODE_LARGEST, the inverse transform will use the largest transform size that fits inside the block. 3) tx_mode=2 for TX_MODE_SELECT, the choice of transform size is specified explicitly for each block. It is the frame-level derived parameters, TxMode. It is derived from the frame-level syntax element, tx_mode_select and the frame level derived parameter Frame Coded Lossless. Default is TxMode=1. Chroma Tx Mode is no longer derived from that of Luma Tx size. Chroma Tx Mode has no explicit setting in the bitstream, and Chroma Tx Mode can be viewed as always equivalent to TX_MODE_LARGEST for the two Chroma planes.
	0	**Reduced Tx Set Used** Format: U1 **Programming Notes** set to 1 to allow the use of a reduced tx set. set to 0 to allow the full tx set to be used for each tx type. It is the frame level syntax element, reduced_tx_set. Encoder Mode- must be 1
8	31:24	**Frame Level Global Motion Invalid Flags** Format: U8 **Description** It indicates to the AV1 HW pipeline (at block level processing) that the result of parsing the frame level global motion parameters of each reference frame is invalid or not. Each bit represents the invalid flag of a reference frame : it is set to 1, if the global motion parameters for the corresponding reference frame is invalid. Otherwise it is set to 0 for valid (default). Frame Level Global Motion Invalid Flags[31] indicates the validity of reference frame ALTREF.

AVP_PIC_STATE

		... Frame Level Global Motion Invalid Flags[25] indicates the validity of reference frame LAST. Frame Level Global Motion Invalid Flags[24] indicates the validity of reference frame INTRA (this bit is reserved and not being used by HW)... This is an intel defined parameter to give HW a hint of frame header parsing result that can simplify the HW design. It takes on the same value as of the global_motion[ALTRF...LAST].invalid flag, which is set inside read_global_motion() function of the reference C model. Valid in decoder only mode
	23	**Reserved (for future expansion of Global Motion Type[7]** Format: MBZ
	22:21	**Global Motion Type[7]** Format: U2 It specifies the global motion type associated with each reference [LAST_FRAME ... ALTREF_FRAME] that can be used to decode the current frame. Set to 0, specifies IDENTITY(requires no additional warp projection parameter) Set to 1, specifies TRANSLATION (requires 2 additional warp projection parameters) Set to 2, specifies ROTZOOM (requires 4 additional warp projection parameters) Set to 3, specifies AFFINE (requires 6additional warp projection parameters) It is the frame level derived parameter gmtype[LAST_FRAME ... ALTREF_FRAME = 1..7], gmtype[0] is not being used.
	20	**Reserved (for future expansion of Global Motion Type[6]** Format: MBZ
	19:18	**Global Motion Type[6]** Format: U2 It specifies the global motion type associated with each reference [LAST_FRAME ... ALTREF_FRAME] that can be used to decode the current frame. Set to 0, specifies IDENTITY(requires no additional projection parameter) Set to 1, specifies TRANSLATION (requires 2 additional projection parameters) Set to 2, specifies ROTZOOM (requires 4 additional projection parameters) Set to 3, specifies AFFINE (requires 6additionalprojection parameters) It is the frame level derived parameter gmtype[LAST_FRAME ... ALTREF_FRAME = 1..7], gmtype[0] is not being used.
	17	**Reserved (for future expansion of Global Motion Type[5]** Format: MBZ
	16:15	**Global Motion Type[5]** Format: U2 It specifies the global motion type associated with each reference [LAST_FRAME ... ALTREF_FRAME] that can be used to decode the current frame. Set to 0, specifies IDENTITY(requires no additional projection parameter) Set to 1, specifies TRANSLATION (requires 2 additional projection parameters)

AVP_PIC_STATE		
		Set to 2, specifies ROTZOOM (requires 4 additional projection parameters) Set to 3, specifies AFFINE (requires 6additionalprojection parameters) It is the frame level derived parameter gmtype[LAST_FRAME ... ALTREF_FRAME = 1..7], gmtype[0] is not being used.
	14	**Reserved (for future expansion of Global Motion Type[4]** Format: MBZ
	13:12	**Global Motion Type[4]** Format: U2 It specifies the global motion type associated with each reference [LAST_FRAME ... ALTREF_FRAME] that can be used to decode the current frame. Set to 0, specifies IDENTITY(requires no additional projection parameter) Set to 1, specifies TRANSLATION (requires 2 additional projection parameters) Set to 2, specifies ROTZOOM (requires 4 additional projection parameters) Set to 3, specifies AFFINE (requires 6additionalprojection parameters) It is the frame level derived parameter gmtype[LAST_FRAME ... ALTREF_FRAME = 1..7], gmtype[0] is not being used.
	11	**Reserved (for future expansion of Global Motion Type[3]** Format: MBZ
	10:9	**Global Motion Type[3]** Format: U2 It specifies the global motion type associated with each reference [LAST_FRAME ... ALTREF_FRAME] that can be used to decode the current frame. Set to 0, specifies IDENTITY(requires no additional projection parameter) Set to 1, specifies TRANSLATION (requires 2 additional projection parameters) Set to 2, specifies ROTZOOM (requires 4 additional projection parameters) Set to 3, specifies AFFINE (requires 6additionalprojection parameters) It is the frame level derived parameter gmtype[LAST_FRAME ... ALTREF_FRAME = 1..7], gmtype[0] is not being used.
	8	**Reserved (for future expansion of Global Motion Type[2]** Format: MBZ
	7:6	**Global Motion Type[2]** Format: U2 It specifies the global motion type associated with each reference [LAST_FRAME ... ALTREF_FRAME] that can be used to decode the current frame. Set to 0, specifies IDENTITY(requires no additional projection parameter) Set to 1, specifies TRANSLATION (requires 2 additional projection parameters) Set to 2, specifies ROTZOOM (requires 4 additional projection parameters) Set to 3, specifies AFFINE (requires 6additionalprojection parameters) It is the frame level derived parameter gmtype[LAST_FRAME ... ALTREF_FRAME = 1..7], gmtype[0] is not being used.
	5	**Reserved (for future expansion of Global Motion Type[1]** Format: MBZ

AVP_PIC_STATE

	4:3	**Global Motion Type[1]** Format: | U2 It specifies the global motion type associated with each reference [LAST_FRAME ... ALTREF_FRAME] that can be used to decode the current frame. Set to 0, specifies IDENTITY(requires no additional projection parameter) Set to 1, specifies TRANSLATION (requires 2 additional projection parameters) Set to 2, specifies ROTZOOM (requires 4 additional projection parameters) Set to 3, specifies AFFINE (requires 6additionalprojection parameters) It is the frame level derived parameter gmtype[LAST_FRAME ... ALTREF_FRAME = 1..7], gmtype[0] is not being used.
	2:0	**Reserved (for future expansion of Global Motion Type[0])** Format: | MBZ
9..29	671:0	**Warp Parameters Array [Reference=1 to 7][ProjectionCoeff=0 to 5]** It specifies the Warp Parameter set for each of the 7 reference frames [LAST_FRAME .. ALTREF_FRAME] Each Warp Parameter set contains 6 warp projection coefficient [projection_coeff = 0 to 5] Each projection coefficient is a 16-bit signed integer (2's component). Total 7 references * 6 proj coeff / 2 = 21 Dwords. Different Global Motion Type is reading different number of projection coefficients from the bitstream. All projection coefficients are coded with signed_subexp_with_ref(). The number of bits read for each projection coefficients from the bitstream, depends on the Global Motion Type too. After decoded the signed_subexp_with_ref(), the projection coefficents are further upshifted and round to the final precision. Allowed range for each coeff is [-4096, 4096] for Warp Motion. For translation-only motion type, the max range for each coeff is [-512 to +512]. All the upper bits are assumed to have sign extended. For IDENTITY motion type, all the coefficients should set to 0. Dword 9 15:0 - Warp Parameters Array[1][0] Dword 9 31:16 -Warp Parameters Array[1][1] Dword 10 15:0 -Warp Parameters Array[1][2] Dword 10 31:16-Warp Parameters Array[1][3] ... ... Dword 29 15:0 -Warp Parameters Array[7][4] Dword 29 31:16-Warp Parameters Array[7][5]
30	31:0	**Reserved** Format: | MBZ
31	31:16	**Intra Frame Height In Pixel Minus 1** Format: | U16 Specifies the height of the INTRA picture(Reference Picture0) in pixels. The INTRA picture height in units of luma samples equals (INTRA_FRAME_HeightInPixelMinus1+ 1)

AVP_PIC_STATE

		AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383].
	15:0	**Intra Frame Width In Pixel Minus 1** Format: | U16 Specifies the width of the INTRA picture(Reference Picture0) in pixels. The INTRA picture widthin units of luma samples equals (INTRA_FRAME_WidthInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383].
32	31:16	**Last Frame Height In Pixel Minus 1** Format: | U16 Specifies the height of the LAST picture(Reference Picture1) in pixels. The LAST picture height in units of luma samples equals (LASTFRAME_HeightInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383].
	15:0	**Last Frame Width In Pixel Minus 1** Format: | U16 Specifies the width of the LAST picture(Reference Picture1) in pixels. The LAST picture widthin units of luma samples equals (LASTFRAME_WidthInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383].
33	31:16	**Last2 Frame Height In Pixel Minus 1** Format: | U16 Specifies the height of the LAST2(Reference Picture 2) picture in pixel. The LAST2 picture height in units of luma samples equals (LAST2FRAME_HeightInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383]. For detail description on frame size and programming notes, refer to the bitfieldFrame Height In Pixel Minus 1 and Frame Width In Pixel Minus 1.
	15:0	**Last2 Frame Width In Pixel Minus 1** Format: | U16 Specifies the width of the LAST2 picture(Reference Picture 2) in pixels. The LAST2 picture widthin units of luma samples equals (LAST2FRAME_WidthInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383]. For detail description on frame size and programming notes, refer to the bitfieldFrame Height In Pixel Minus 1 and Frame Width In Pixel Minus 1.
34	31:16	**Last3 Frame Height In Pixel Minus 1** Format: | U16

AVP_PIC_STATE

		Specifies the height of the LAST3 picture(Reference Picture 3) in pixels. The LAST3 picture height in units of luma samples equals (LAST3FRAME_HeightInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383]. For detail description on frame size and programming notes, refer to the bitfieldFrame Height In Pixel Minus 1 and Frame Width In Pixel Minus 1.
	15:0	**Last3 Frame Width In Pixel Minus 1** Format: U16 Specifies the width of the LAST3 picture(Reference Picture 3) in pixels. The LAST3 picture widthin units of luma samples equals (LAST3FRAME_WidthInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383]. For detail description on frame size and programming notes, refer to the bitfieldFrame Height In Pixel Minus 1 and Frame Width In Pixel Minus 1.
35	31:16	**Golden Frame Height In Pixel Minus 1** Format: U16 Specifies the height of the GOLDEN picture(Reference Picture 4) in pixels. The GOLDEN picture height in units of luma samples equals (GOLDENFRAME_HeightInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383].
	15:0	**Golden Frame Width In Pixel Minus 1** Format: U16 Specifies the width of the GOLDEN picture(Reference Picture 4) in pixels. The GOLDEN picture widthin units of luma samples equals (GOLDENFRAME_WidthInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383].
36	31:16	**BWDREF Frame Height In Pixel Minus 1** Format: U16 Specifies the height of the BWDREF picture(Reference Picture 5) in pixels. The BWDREF picture height in units of luma samples equals (BWDREFFRAME_HeightInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383].
	15:0	**BWDREF Frame Width In Pixel Minus 1** Format: U16 Specifies the width of the BWDREF picture(Reference Picture 5) in pixels. The BWDREF picture widthin units of luma samples equals (BWDREFFRAME_WidthInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size.

AVP_PIC_STATE		
		Valid Value = [15, 16383].
37	31:16	**ALTREF2 Frame Height In Pixel Minus 1** Format: U16 Specifies the height of the ALTREF2 picture(Reference Picture 6) in pixels. The ALTREF picture height in units of luma samples equals (ALTREFFRAME_HeightInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383]. For detail description on frame size and programming notes, refer to the bitfieldFrame Height In Pixel Minus 1 and Frame Width In Pixel Minus 1.
	15:0	**ALTREF2 Frame Width In Pixel Minus 1** Format: U16 Specifies the width of the ALTREF2 picture(Reference Picture 6) in pixels. The ALTREF picture widthin units of luma samples equals (ALTREFFRAME_WidthInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383]. For detail description on frame size and programming notes, refer to the bitfieldFrame Height In Pixel Minus 1 and Frame Width In Pixel Minus 1.
38	31:16	**ALTREF Frame Height In Pixel Minus 1** Format: U16 Specifies the height of the ALTREF picture(Reference Picture 7) in pixels. The ALTREF2 picture height in units of luma samples equals (ALTREF2FRAME_HeightInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383]. For detail description on frame size and programming notes, refer to the bitfieldFrame Height In Pixel Minus 1 and Frame Width In Pixel Minus 1.
	15:0	**ALTREF Frame Width In Pixel Minus 1** Format: U16 Specifies the width of the ALTREF picture(Reference Picture 7) in pixels. The ALTREF2 picture widthin units of luma samples equals (ALTREF2FRAME_WidthInPixelMinus1+ 1) AV1 supports up to 64K frame size. Intel supports up to 16K frame size. Valid Value = [15, 16383]. For detail description on frame size and programming notes, refer to the bitfieldFrame Height In Pixel Minus 1 and Frame Width In Pixel Minus 1.
39	31:16	**Horizontal Scale Factor for INTRA** Format: U2.14 This indicates the scaling factor between current frame and the INTRA reference frame (Reference Picture0). Set to (INTRA_Width * 2^14 + (CurrentWidth / 2)) / CurrentWidth Scaling Factor can be [1/16 to 2].

AVP_PIC_STATE

	15:0	**Vertical Scale Factor for INTRA** Format: U2.14 This indicates the scaling factor between current frame and the INTRA reference frame (Reference Picture 0). Set to (INTRA_Height * 2^14 + (CurrentHeight / 2)) / CurrentHeight Scaling Factor can be [1/16 to 2].
40	31:16	**Horizontal Scale Factor for LAST** Format: U2.14 This indicates the scaling factor between current frame and the LAST reference frame (Reference Picture1). Set to (LAST_Width * 2^14 + (CurrentWidth / 2)) / CurrentWidth Scaling Factor can be [1/16 to 2].
	15:0	**Vertical Scale Factor for LAST** Format: U2.14 This indicates the scaling factor between current frame and the LAST reference frame (Reference Picture 1). Set to (LAST_Height * 2^14 + (CurrentHeight / 2)) / CurrentHeight Scaling Factor can be [1/16 to 2].
41	31:16	**Horizontal Scale Factor for LAST2** Format: U2.14 This indicates the scaling factor between current frame and the LAST2 reference frame (Reference Picture 2). Set to (LAST2_Width * 2^14 + (CurrentWidth / 2)) / CurrentWidth Scaling Factor can be [1/16 to 2].
	15:0	**Vertical Scale Factor for LAST2** Format: U2.14 This indicates the scaling factor between current frame and the LAST2 reference frame (Reference Picture 2). Set to (LAST2_Height * 2^14 + (CurrentHeight / 2)) / CurrentHeight Scaling Factor can be [1/16 to 2].
42	31:16	**Horizontal Scale Factor for LAST3** Format: U2.14 This indicates the scaling factor between current frame and the LAST3 reference frame (Reference Picture 3). Set to (LAST3_Width * 2^14 + (CurrentWidth / 2)) / CurrentWidth Scaling Factor can be [1/16 to 2].
	15:0	**Vertical Scale Factor for LAST3** Format: U2.14 This indicates the scaling factor between current frame and the LAST3 reference frame (Reference Picture 3). Set to (LAST3_Height *2^14 + (CurrentHeight / 2)) / CurrentHeight Scaling Factor can be [1/16 to 2].

AVP_PIC_STATE

43	31:16	**Horizontal Scale Factor for GOLDEN** Format: U2.14 This indicates the scaling factor between current frame and the GOLDEN reference frame (Reference Picture 4). Set to (GOLDEN_Width * 2^14 + (CurrentWidth / 2)) / CurrentWidth Scaling Factor can be [1/16 to 2].
	15:0	**Vertical Scale Factor for GOLDEN** Format: U2.14 This indicates the scaling factor between current frame and the GOLDEN reference frame (Reference Picture 4). Set to (GOLDEN_Height * 2^14 + (CurrentHeight / 2)) / CurrentHeight Scaling Factor can be [1/16 to 2].
44	31:16	**Horizontal Scale Factor for BWDREF_FRAME** Format: U2.14 This indicates the scaling factor between current frame and the BWDREF_FRAME reference frame (Reference Picture 5). Set to (BWDREF_FRAME_Width * 2^14 + (CurrentWidth / 2)) / CurrentWidth Scaling Factor can be [1/16 to 2].
	15:0	**Vertical Scale Factor for BWDREF_FRAME** Format: U2.14 This indicates the scaling factor between current frame and the BWDREF_FRAME reference frame (Reference Picture 5). Set to (BWDREF_FRAME_Height * 2^14 + (CurrentHeight / 2)) / CurrentHeight Scaling Factor can be [1/16 to 2].
45	31:16	**Horizontal Scale Factor for ALTREF2** Format: U2.14 This indicates the scaling factor between current frame and the ALTREF2 reference frame (Reference Picture 6). Set to (ALTREF_FRAME_Width * 2^14 + (CurrentWidth / 2)) / CurrentWidth Scaling Factor can be [1/16 to 2].
	15:0	**Vertical Scale Factor for ALTREF2** Format: U2.14 This indicates the scaling factor between current frame and the ALTREF2 reference frame(Reference Picture 6). Set to (ALTREF_FRAME_Height * 2^14 + (CurrentHeight / 2)) / CurrentHeight Scaling Factor can be [1/16 to 2].
46	31:16	**Horizontal Scale Factor for ALTREF** Format: U2.14 This indicates the scaling factor between current frame and the ALTREFreference frame (Reference Picture 7). Set to (ALTREF2_FRAME_Width * 2^14 + (CurrentWidth / 2)) / CurrentWidth Scaling Factor can be [1/16 to 2].

AVP_PIC_STATE		
	15:0	**Vertical Scale Factor for ALTREF** Format: U2.14 This indicates the scaling factor between current frame and the ALTREFreference frame(Reference Picture 7). Set to (ALTREF2_FRAME_Height * 2^14 + (CurrentHeight / 2)) / CurrentHeight Scaling Factor can be [1/16 to 2].
47	31:24	**Reference Frame Order Hint [3] for Last3 Frame** Format: U8 It specifies the expected output order hint for each reference buffer. This is the LAST3 Reference Frame Order Hint (ReferenceFrame 3). Note : The values in the ref_order_hint array are provided to allow implementations to gracefully handle cases when some frames have been lost.
	23:16	**Reference Frame Order Hint [2] for Last2 Frame** Format: U8 It specifies the expected output order hint for each reference buffer. This is the LAST2 Reference Frame Order Hint (ReferenceFrame 2). Note : The values in the ref_order_hint array are provided to allow implementations to gracefully handle cases when some frames have been lost.
	15:8	**Reference Frame Order Hint [1] for Last Frame** Format: U8 It specifies the expected output order hint for each reference buffer. This is the LAST Reference Frame Order Hint (ReferenceFrame 1). Note : The values in the ref_order_hint array are provided to allow implementations to gracefully handle cases when some frames have been lost.
	7:0	**Reference Frame Order Hint [0] for Intra Frame** Format: U8 It specifies the expected output order hint for each reference buffer. This is the INTRA Reference Frame Order Hint (ReferenceFrame 0). It is a derived frame-level parameter, which can be equal to 1) the frame-level syntax element, ref_order_hint[i=0 to 7], which is only present in the bitstream if the current frame type (frame-level syntax)is NOT a KEY Frame, AND frame level syntax element: error-resilient mode is set to 1 AND sequence level syntax element: enable_order_hint is set to 1. 2) OR, the saved order hint, when the reference frame was the current frame being decoded. Note : The values in the ref_order_hint array can be used to implement to gracefully handle cases when some frames have been lost. It is done at the SW level, not inside AVP HW pipeline.
48	31:24	**Reference Frame Order Hint [7] for ALTREF Frame** Format: U8 It specifies the expected output order hint for each reference buffer. This is the ALTREF Reference Frame Order Hint (ReferenceFrame 7).

<table>
<tr><th colspan="3">AVP_PIC_STATE</th></tr>
<tr><td></td><td></td><td>Note : The values in the ref_order_hint array are provided to allow implementations to gracefully handle cases when some frames have been lost.</td></tr>
<tr><td></td><td>23:16</td><td><b>Reference Frame Order Hint [6] for ALTREF2 Frame</b><table><tr><td>Format:</td><td>U8</td></tr></table>It specifies the expected output order hint for each reference buffer.
This is the ALTREF2 Reference Frame Order Hint (ReferenceFrame 6).
Note : The values in the ref_order_hint array are provided to allow implementations to gracefully handle cases when some frames have been lost.</td></tr>
<tr><td></td><td>15:8</td><td><b>Reference Frame Order Hint [5] for BWDREF Frame</b><table><tr><td>Format:</td><td>U8</td></tr></table>It specifies the expected output order hint for each reference buffer.
This is the BWDREF Reference Frame Order Hint (ReferenceFrame 5).
Note : The values in the ref_order_hint array are provided to allow implementations to gracefully handle cases when some frames have been lost.</td></tr>
<tr><td></td><td>7:0</td><td><b>Reference Frame Order Hint [4] for Golden Frame</b><table><tr><td>Format:</td><td>U8</td></tr></table>It specifies the expected output order hint for each reference buffer.
This is the GOLDEN Reference Frame Order Hint (ReferenceFrame 4).
Note : The values in the ref_order_hint array are provided to allow implementations to gracefully handle cases when some frames have been lost.</td></tr>
<tr><td>49</td><td>31:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>50</td><td>31:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>51</td><td>31:28</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>27</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>26</td><td><b>FrameSzUnderStatusEn - FrameBitRateMinReportMask</b><table><tr><td>Format:</td><td>U1</td></tr></table>This is a mask bit controlling if the condition of frame level bit count is less than FrameBitRateMin.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Disable</td><td>Do not update bit 2 (Frame Bit Count Violate -- under run) of HCP_VP9_IMAGE_STATUS control register.</td></tr><tr><td>1</td><td>Enable</td><td>Set bit 2 (Frame Bit Count Violate -- under run) of</td></tr></table></td></tr>
</table>

AVP_PIC_STATE

		HCP_VP9_IMAGE_STATUS control register if the total frame level bit counter is less than or equal to Frame Bit Rate Minimum limit.

Programming Notes
Encoder Only

25

FrameSzOverStatusEn - FrameBitRateMaxReportMask

Format:	U1

This is a mask bit controlling if the condition of frame level bit count exceeds FrameBitRateMax.

Value	Name	Description
0	Disable	Do not update bit 1 of HCP_VP9_IMAGE_STATUS control register.
1	Enable	Set bit 1 of HCP_VP9_IMAGE_STATUS control register if the total frame level bit counter is greater than or equal to Frame Bit Rate Maximum limit.

Programming Notes
Encoder Only

24:18

Reserved

Access:	RO
Format:	MBZ

17

Reserved

Access:	RO
Format:	MBZ

16

NonFirstPassFlag

This signals the current pass is not the first pass. It will imply designate HW behavior.

Value	Name	Description
0	Disable	If it is initial-Pass, this bit is set to 0.
1	Enable	For subsequent passes, this bit is set to 1.

Programming Notes
Encoder Only

15:0

Reserved

Access:	RO
Format:	MBZ

52

31

FrameBitrateMaxUnit

Format:	U1

This field is the Frame Bitrate Maximum Limit Units.

AVP_PIC_STATE

Value	Name	Description
0	Byte	32byte unit
1	KiloByte	4Kbyte unit

Programming Notes
Encoder Only

30:14 **Reserved**

Access:	RO
Format:	MBZ

13:0 **FrameBitRateMax**

Format:	U14

This field is the Frame Bitrate Maximum Limit. This field along with FrameBitrateMaxUnit determines maximum allowed bits in a frame before multi-pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count exceeds this value.
0-512KB The programmable range is 0-512KB when FrameBitrateMaxUnit is 0.
0-64MB The programmable range is 0-64Mbyte when FrameBitrateMaxUnit is 1.

Programming Notes
Encoder Only

53

31 **FrameBitrateMinUnit**

Format:	U1

This field is the Frame Bitrate Maximum Limit Units.

Value	Name	Description
0	Byte	32byte unit
1	KiloByte	4Kbyte unit

Programming Notes
Encoder Only

30:14 **Reserved**

Access:	RO
Format:	MBZ

13:0 **FrameBitRateMin**

Format:	U14

This field is the Frame Bitrate Minimum Limit. This field along with FrameBitrateMinUnit determines maximum allowed bits in a frame before multi-pass gets triggered (when enabled). In other words, multi-pass is triggered when

AVP_PIC_STATE

		the actual frame byte count exceeds this value. 0-512KB The programmable range is 0-512KB when FrameBitrateMinUnit is 0. 0-64MB The programmable range is 0-64Mbyte when FrameBitrateMinUnit is 1. **Programming Notes** Encoder Only
54..55	63:0	**FrameDeltaQindexMax** Format: ExtendedMessageDescriptor-SamplingEngineNon-Bindless Frame level delta Qindex which should be used in case FrameSize - FrameBitRateMax in the range of ((FrameDeltaQindexLFMaxRange[n] * FrameBitRateMax>>5)), FrameDeltaQindexLFMaxRange[n+1] * FrameBitRateMax>>5)). Each DelatQindexMax value is 8-bit with S7M format **Programming Notes** If n == 7, DeltaQpMaxRange is infinity. Encoder Only
56	31:0	**FrameDeltaQindexMin** Format: ExtendedMessageDescriptor-SamplingEngineNon-Bindless Frame level delta Qindex which should be used in case FrameSize - FrameBitRateMin in the range of ((FrameDeltaQindexLFMinRange[n] * FrameBitRateMin>>5)), FrameDeltaQindexLFMinRange[n+1] * FrameBitRateMin>>5)). Each DelatQindexMin value is 8-bit with S7M format **Programming Notes** If n == 3, FrameDeltaQindexLFMaxRange is zero. (n>3 is not supported) Encoder Only
57..58	63:0	**FrameDeltaLFMax** Format: ExtendedMessageDescriptor-SamplingEngineNon-Bindless Frame level delta Loop Filter Level which should be used in case FrameSize - FrameBitRateMax in the range of ((FrameDeltaQindexLFMaxRange[n] * FrameBitRateMax>>5)), FrameDeltaQindexLFMaxRange[n+1] * FrameBitRateMax>>5)). Each delta_lf_max is 7 bits with S6M format **[bits 7, 15, 23, 31,.63 are reserved]** **Programming Notes** If n == 7, FrameDeltaQindexLFMaxRange is infinity. Encoder Only
59	31:0	**FrameDeltaLFMin** Format: ExtendedMessageDescriptor-SamplingEngineNon-Bindless Frame level delta Loop Filter Level which should be used in case FrameSize -

<table>
<tr><th colspan="3">AVP_PIC_STATE</th></tr>
<tr><td></td><td></td><td>FrameBitRateMin in the range of ((FrameDeltaQindexLFMinRange[n] * FrameBitRateMin>>5)), FrameDeltaQindexLFMinRange[n+1] * FrameBitRateMin>>5)).
Each delta_lf_min is 7 bits with S6M format
<b>[bits 7, 15, 23, 31 are reserved]</b>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>If n == 3, FrameDeltaQindexLFMaxRange is zero. (n>3 is not supported)</td></tr>
<tr><td>Encoder Only</td></tr>
</table></td></tr>
<tr><td>60..61</td><td>63:0</td><td><b>FrameDeltaQindexLFMaxRange</b>
Condition: FrameDeltaQindexLFMaxRange[n] >= FrameDeltaQindexLFMaxRange[n-1] This field is to calculate ranges for Frame level delta Qindex, specifically Frame level delta Qindex[n] and Frame level delta Qindex[n+1].
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>If n == 0, FrameDeltaQindexLFMaxRange is zero.</td></tr>
<tr><td>Encoder Only</td></tr>
</table></td></tr>
<tr><td>62</td><td>31:0</td><td><b>FrameDeltaQindexLFMinRange</b>
Condition: FrameDeltaQindexLFMinRange[n] >= FrameDeltaQindexLFMinRange[n-1]This field is to calculate ranges for Frame level delta Qindex, specifically Frame level delta Qindex[n] and Frame level delta Qindex[n+1].
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>If n == 0, FrameDeltaQindexLFMinRange is zero.</td></tr>
<tr><td>Encoder Only</td></tr>
</table></td></tr>
<tr><td>63</td><td>31:30</td><td><b>MinFrameSizeUnits</b>
<table>
<tr><td>Format:</td><td>U2</td></tr>
</table>
This field is the Minimum Frame Size Units
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>4Kb</td><td>Minimum Frame Size is in 4Kbytes.</td></tr>
<tr><td>1</td><td>16Kb</td><td>Minimum Frame Size is in 4Kbytes.</td></tr>
<tr><td>2</td><td>Comaptibility Mode</td><td></td></tr>
<tr><td>3</td><td>6 Bytes</td><td></td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Encoder Only</td></tr>
</table></td></tr>
<tr><td></td><td>29:16</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>15:0</td><td><b>MinFramSize</b>
<table>
<tr><td>Format:</td><td>U16</td></tr>
</table>
Minimum Frame Size [15:0] (in Word, 16-bit)(Encoder Only) Minimum Frame Size is</td></tr>
</table>

AVP_PIC_STATE		
		specified to compensate for intel Rate Control Currently zero fill (no need to perform emulation byte insertion) is done at the last slice of a picture. It is needed for CBR. Intel encoder parameter, not part of DXVA. The caller should always make sure that the value, represented by Minimum Frame Size, is always less than maximum frame size FrameBitRateMax. This field is reserved in Decode mode. **Programming Notes** Programmable range is 0..(2^16-1) * 2^12 when MinFrameSizeUnits is 0. (4KB unit) Programmable range is 0..(2^16-1) * 2^14 when MinFrameSizeUnits is 1. (16KB unit) Encoder Only
64	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Reserved MBZ** Format: U16
65	31:16	**Class0_SSE_Threshold1** Format: U16 **Programming Notes** This field specifies the upper bound threshold for Class0 Zone1 to classify the per-4x4sblk SSE statistics. Class0_SSE_Threshold_0 < per-4x4sblk SSE <= Class0_SSE_Threshold_1 fall under Class0 Zone1. Class0_SSE_Threshold_1 < per-4x4sblk SSE fall under Class0 Zone2. Encoder Only
	15:0	**Class0_SSE_Threshold0** Format: U16 **Programming Notes** This field specifies the upper bound threshold for Class0 Zone0 to classify the per-4x4sblk SSE statistics. per-4x4sblk SSE <= Class0_SSE_Threshold_0 fall under Class0 Zone0. Encoder Only
66..73 **Programming Notes:** SSE thresholds for Class 1-8, see DW 33 (SSE Class 0 thresholds) for format.	255:0	**SSE thresholds for Class1-8** Format: U256

AVP_PIC_STATE		
Encoder Only		
74	31:0	**rdmult** rdmult is used in Wiener Filter search algorithm and its' derived as, *rdmult = 88 * q * q / 24; (for 8bits)* rdmult = ROUND_POWER_OF_TWO(88 * q * q / 24, 4); (for 10 bits) rdmult = ROUND_POWER_OF_TWO(88 * q * q / 24, 8); (for 12 bits) Valid in encoder mode only
75	31:0	**Reserved MBZ**

AVP_PIPE_BUF_ADDR_STATE

AVP_PIPE_BUF_ADDR_STATE

Source:	VideoCS
Length Bias:	2

The AVP Pipeline is selected with the **Media Instruction Opcode "8h"** for all AVP Commands. Each AVP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This state command provides the physical memory base addresses for all row store buffers, column store buffers, reconstructed output and reference frame buffers, and auxiliary data buffers (MV, segment map, etc.) that are required by the AV1 decoding and encoding process.
This is a frame level state command and is shared by both encoding and decoding processes.
AVP is a tile based pipeline and is a stateless pipeline, hence all sequence level, frame level, and segment level state commands must be resent to process each tile.
Memory compression may be applicable to some of these buffers for BW saving.
Note : there is no buffer to store the 16 QM table sets, they are implemented directly inside the HW pipeline.

DWord	Bit	Description		
0	31:29	**Command Type**		
			Default Value:	3h PARALLEL_VIDEO_PIPE
			Format:	OpCode
	28:27	**Pipeline Type**		
			Default Value:	2h
			Format:	OpCode
	26:23	**Media Instruction Opcode**		
			Default Value:	3h Codec/Engine Name
			Format:	OpCode
			Codec/Engine Name = AVP = 3h	
	22:16	**Media Instruction Command**		
			Default Value:	2h AVP_PIPE_BUF_ADDR_STATE
			Format:	OpCode
	15:12	**Reserved**		
			Access:	RO
			Format:	MBZ
	11:0	**Dword Length**		
			Format:	=n
			(Excludes Dwords 0, 1).	
			Value	**Name**
			C0h	
1..16	511:0	**Reference Frame Buffer Base Address (RefAddr[0-7])**		
			Format:	**SplitBaseAddress64ByteAligned[8]**

AVP_PIPE_BUF_ADDR_STATE

		This array specifies the physical memory base addresses of the 8 visible reference frame buffers using for inter-prediction. They all contain previously reconstructed/decoded frames. However, this Reference Frame Buffer array is a remapped version (see below the mapping equations)of the DPB buffer array specified in the AV1 Spec and Reference C Model. Intel implementation has converted the 2 level of reference frame indexing (using RefFrame[0..6] and ref_frame_idx[0..6]) into a single level indexing (using the enumINTRA_FRAME, LAST_FRAME, ..., ALTREF_FRAME, in this fixed order). In the process, the ref_frame_idx[0..6] array is eliminated by intel. Application and Driver will continue to receive or parse the bitstream header based on the2-level indexing AV1 DPB buffer array definition, which can contain more than 8 frame buffers, but only at most 8 of them are visible to the AVP HW pipeline and is in the formed of single level indexing intel-remapped-DPB array. Typically, these reference frame buffers are read-only, for the purpose of Motion Comp. Memory compression is applied in accessing these buffers. At most only 7 out of the 8 visible reference frames can be used for Motion Comp.in decoding inter-coded blocks of the current frame. The subset of reference frames being used in decoding the current frame is setup by the Application and Driver. But it is recommended to set all Reference Frame Buffer Base Address to valid and known addresses for error handling. AV1 Reference Frames are defined in the following order: DW 0-1 RefAddr[0] - INTRA Frame (Reference Frame 0) DW 2-3 RefAddr[1] - LAST Frame (Reference Frame 1) In AV1 Spec, it is mapped into DPB [ref_frame_idx[LAST_FRAME-LAST_FRAME]] DW 4-5 RefAddr[2] - LAST2 Frame (Reference Frame 2) In AV1 Spec, it is mapped into DPB [ref_frame_idx[LAST2_FRAME-LAST_FRAME]] DW 6-7 RefAddr[3] - LAST3 Frame (Reference Frame 3) In AV1 Spec, it is mapped into DPB [ref_frame_idx[LAST3_FRAME-LAST_FRAME]] DW 8-9 RefAddr[4] - GOLDEN Frame (Reference Frame 4) In AV1 Spec, it is mapped into DPB [ref_frame_idx[GOLDEN_FRAME-LAST_FRAME]] DW 10-11RefAddr[5] - BWDREF Frame (Reference Frame 5)In AV1 Spec, it is mapped into DPB [ref_frame_idx[BWDREF_FRAME-LAST_FRAME]] DW 12-13RefAddr[6] - ALTREF2 Frame (Reference Frame 6)In AV1 Spec, it is mapped into DPB [ref_frame_idx[ALTREF2_FRAME-LAST_FRAME]] DW 14-15RefAddr[7] - ALTREF Frame (Reference Frame 7) In AV1 Spec, it is mapped into DPB [ref_frame_idx[ALTREF_FRAME-LAST_FRAME]] Note : for 48 bit address, a pair of Dwords (i.e. 2) are needed to store each base address. Note : This reference frame naming convention is a legacy specification in the reference C model. Although the very first reference frame is labeled as INTRA, it is not reserved only for a previously decoded KEY frame or a decoded INTRA-ONLY NON-KEY frame. Any newly decoded frame can be stored in any one of these 8 reference frame buffers for future reference - it is up to the Application and Driver that handle the DPB management according to the bitstream and normative rules specified in AV1. Note : When IntraBC coding mode is used in the KEY Frame or in the INTRA-ONLY NON-KEY Frame, the intraBC coded block is decoded as a regular inter-coded block, and one of these 8 reference frame buffers will be directly used in a READ/WRITE fashion. ??? Note : the format of the pixels (Y, U and V components)stored inside these reference frame buffers is defined in the AV1_Surface_State Command. For monochrome video, all reference frame buffers can only have Luma Y component.

AVP_PIPE_BUF_ADDR_STATE		
		Note :All reference frame buffers' surface addresses must be 4K byte aligned. There is a max. of 8 Reference Picture Buffer Addresses, and all share the same third address DW in specifying 48-bit address.
17	31:0	**Reference Frame Buffer Base Address Attributes** Format: **MemoryAddressAttributes** All reference frame buffers' surface addresses must be 4K byte aligned. There is a max. of 8 Reference Picture Buffer Addresses, and all share this same third address DW in specifying 48-bit address.
18..19	63:0	**Decoded Output Frame Buffer Address** Format: **SplitBaseAddress4KByteAligned** It specifies the physical memory base addresses of the frame buffer that stores the final pixel output of the AVP pipeline, just before the Film Grain unit(could be sent for display). Typically, it is a WRITE-only surface with pixel format specified in the AV1_SURFACE_STATE. It can be 4:0:0, 4:2:0, 4:2:2 or 4:4:4. Memory compression is applied to this surface. This buffer holds the result of reconstructing the current frame, including all the Post In-Loop Filtering Processes (Deblocker, CDEF, Super-Resolution, and/or Loop Restoration) that are enabled. There can be a few actions to follow: 1) Application and Driver will then place this newly decoded frame into the Display Buffer (DPB), as one of the 8 reference frames, only if it will be used for the decoding of a later frame(s) (??? a bit in the frame header is set to indicate this ???). 2) At the same time, if this newly decoded frame is to be displayed immediately, it will also be copied and sent to other system for further processing or directly to the Display Controller. 3) If Film Grain Injection is enabled in the sequence header, this newly decoded frame is further processed by the Out of Loop Film Grain Synthesis unit in a block-based pipeline fashion (continued from the block-based pixel reconstruction pipeline of the AV1 decoder). Note: when intraBC is active, it can also be a READ surface for the Motion Comp operation. ??? Note : there is only one write location along the in-loop decoding pipeline, and it is at the output of the Loop Restoration Filter. If any or all of the Post In-Loop Filters (Deblocker, CDEF, Super-Resolution, Loop Restoration)are disabled, the reconstructed pixels still need to pipethrough those disabled filters without change (i.e. simply bypass), until they reach the output of the Loop Restoration Filter. Note : if it is decided to implement some of the Post In-Loop Filters in software (as a separate pass - not being implemented yet), this buffer will hold the final pixel output frame of the HW pipeline. Note : this decoded output buffer if going to be used as a reference frame, then it is added into the Display Buffer (DPB) at the location specified by the frame level syntax element refresh_frame_flags. If the current frame is a KEY Frame or a SWITCH Frame, refresh_frame_flags is set to all 1's, so the entire DPB is initialized (KEY Frame)/re-initialized (SWITCH Frame) to the current frame.
20	31:0	**Decoded Output Frame Buffer Address Attributes** Format: **MemoryAddressAttributes**

AVP_PIPE_BUF_ADDR_STATE

21..23	95:0	**Reserved** Access: RO Format: MBZ
24..25	63:0	**IntraBC Decoded Output Frame Buffer Address** Format: SplitBaseAddress4KByteAligned When IntraBC (frame level) is ON, the current decoded and reconstructed partial pixel frame is needed for motion compensation. The normal Decoder Output Frame Buffer is typically written out with memory compression ON, as such it is not suitable to be read back for performing motion compensation within the decoding of the same frame. A separate decoded output frame buffer is needed, whose surface buffer does not have memory compression turned ON. That is, when IntraBC is ON (frame level), AVP will write out two decoded output frame with identical content.
26	31:0	**IntraBC Decoded Output Frame Buffer Address Attributes** Format: MemoryAddressAttributes
27..28	63:0	**CDF Tables Initialization Buffer Address** Format: SplitBaseAddress64ByteAligned Base address for the CDF Tables Initialization Buffer. This Buffer is read-only. It is programmable with the initial CDF table set for the entire frame and for all its tiles at the very beginning before any decoding process. The content of this buffer must be the same for all tiles for the frame
29	31:0	**CDF Tables Initialization Buffer Address Attributes** Format: MemoryAddressAttributes
30..31	63:0	**CDF Tables Backward Adaptation Buffer Address** Format: SplitBaseAddress64ByteAligned Baseaddress of the CDF Tables Backward Adaptation Buffer. This Buffer stores the updated frame context of the largest tile in the current decoded frame. This is a WRITE-Only buffer.
32	31:0	**CDF Tables Backward Adaptation Buffer Address Attributes** Format: MemoryAddressAttributes
33..34	63:0	**AV1 Segment ID Read Buffer Address** Format: SplitBaseAddress64ByteAligned Specifies the 64 byte aligned buffer address for AV1 SegmentID buffer. This should contain the writeout SegmentID from previous frame and will be used to predict SegmentID for the current frame. Hardware will write out SegmentID of the current frame in the same address for the next frame. It is used for temporal prediction of segment ID in the current frame. The segment map has a granularity of 4x4 blocks.
35	31:0	**AV1 Segment ID Read Buffer Address Attributes** Format: MemoryAddressAttributes
36..37	63:0	**AV1 Segment ID Write Buffer Address**

AVP_PIPE_BUF_ADDR_STATE

		Format: SplitBaseAddress64ByteAligned Specifies the 64 byte aligned buffer address for AV1 SegmentID buffer. This should contain the writeout SegmentID of the current frame and will be used to predict SegmentID for later frame. This segment map buffer is attached to the current decoded frame as its auxiliary data, and are both stored together in the Display Buffer (DPB), if the current frame is to be used as a reference frame to decode later frame(s). The segment map has a granularity of 4x4 blocks.
38	31:0	**AV1 Segment ID Write Buffer Address Attributes** Format: MemoryAddressAttributes
39..54	511:0	**Collocated Motion Vector Temporal Buffer Base Address (TmvAddr[0-7])** Format: SplitBaseAddress64ByteAligned[8] Base address for the Collocated Motion Vector Temporal buffer. The 8 Temporal Buffersare defined in the following order: DW 38-39 TmvAddr[0] - INTRA Frame (Reference Frame 0) DW 40..41 TmvAddr1] - LAST Frame (Reference Frame 1) DW 42..43 TmvAddr[2] - LAST2 Frame (Reference Frame 2) DW 44..45 TmvAddr[3] - LAST3 Frame (Reference Frame 3) DW 46..47 TmvAddr4] - GOLDEN Frame (Reference Frame 4) DW 48..49 TmvAddr[5] - BWDREF Frame (Reference Frame 5) DW 50..51 TmvAddr[6] - ALTREF2 Frame (Reference Frame 6) DW 52..53 TmvAddr[7] - ALTREF Frame (Reference Frame 7) Note : There is a max. of 8 Collocated MV TemporalBuffer Addresses, and all share the same third address DW in specifying 48-bit address
55	31:0	**Collocated Motion Vector Temporal Buffer Base Address Attributes** Format: MemoryAddressAttributes Note : There is a max. of 8 Collocated MV TemporalBuffer Addresses, and all share this same third address DW in specifying 48-bit address.
56..57	63:0	**Current Frame Motion Vector Write Buffer Address** Format: SplitBaseAddress64ByteAligned Base address for the Current Motion Vector Temporal buffer.
58	31:0	**Current Frame Motion Vector Write Buffer Address Attributes** Format: MemoryAddressAttributes
59..61	95:0	**Reserved** Access: RO Format: MBZ
62..63	63:0	**Bitstream Decoder/Encoder Line Rowstore Read/Write Buffer Address** Format: SplitBaseAddress64ByteAligned Specifies the 64 byte aligned buffer address for Bitstream Decode Line Rowstore
64	31:0	**Bitstream Decoder/Encoder Line Rowstore Read/Write Buffer Address Attributes** Format: MemoryAddressAttributes

AVP_PIPE_BUF_ADDR_STATE

65..66	63:0	**Bitstream Decoder/Encoder Tile Line Rowstore Read/Write Buffer Address** Format: **SplitBaseAddress64ByteAligned** Specifies the 64 byte aligned buffer address for Bitstream Decode Tile Line Buffer
67	31:0	**Bitstream Decoder/Encoder Tile Line Rowstore Read/Write Buffer Address Attributes** Format: **MemoryAddressAttributes**
68..69	63:0	**Intra Prediction Line Rowstore Read/Write Buffer Address** Format: **SplitBaseAddress64ByteAligned** Specifies the 64 byte aligned buffer address for Intra Prediction Line Rowstore
70	31:0	**Intra Prediction Line Rowstore Read/Write Buffer Address Attributes** Format: **MemoryAddressAttributes**
71..72	63:0	**Intra Prediction Tile Line Rowstore Read/Write Buffer Address** Format: **SplitBaseAddress64ByteAligned** Specifies the 64 byte aligned buffer address for Intra Prediction Tile Line Rowstore
73	31:0	**Intra Prediction Tile Line Rowstore Read/Write Buffer Address Attributes** Format: **MemoryAddressAttributes**
74..75	63:0	**Spatial Motion Vector Line Read/Write Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base address for the Spatial Motion Vector Line buffer.
76	31:0	**Spatial Motion Vector Line Read/Write Buffer Address Attributes** Format: **MemoryAddressAttributes**
77..78	63:0	**Spatial Motion Vector Coding Tile Line Read/Write Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base address for the Spatial Motion Vector Tile Line buffer.
79	31:0	**Spatial Motion Vector Tile Line Read/Write Buffer Address Attributes** Format: **MemoryAddressAttributes**
80..81	63:0	**Loop Restoration Meta Tile Column Read/Write Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base Address forLoop Restoration Meta Tile Column Read/Write Buffer
82	31:0	**Loop Restoration Meta Tile Column Read/Write Buffer Address Attributes** Format: **MemoryAddressAttributes**
83..84	63:0	**Loop Restoration Filter Tile Read/Write Line Y Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base address for the Loop Restoration Filter Tile Line Y Buffer
85	31:0	**Loop Restoration Filter Tile Read/Write Line Y Buffer Address Attributes** Format: **MemoryAddressAttributes**
86..87	63:0	**Loop Restoration Filter Tile Read/Write Line U Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base address for the Loop Restoration Filter Tile Line U Buffer

AVP_PIPE_BUF_ADDR_STATE		
88	31:0	**Loop Restoration Filter Tile Read/Write Line U Buffer Address Attributes** Format: MemoryAddressAttributes
89..90	63:0	**Loop Restoration Filter Tile Read/Write Line V Buffer Address** Format: SplitBaseAddress64ByteAligned Base address for the Loop Restoration Filter Tile Line V Buffer
91	31:0	**BitField: Loop Restoration Filter Tile Read/Write Line V Buffer Address Attributes** Format: MemoryAddressAttributes
92..93	63:0	**Deblocker Filter Line Read/Write Y Buffer Address** Format: SplitBaseAddress64ByteAligned Base address of the filter line buffer (read/write) used by the Deblocking Filter.
94	31:0	**Deblocker Filter Line Read/Write Y Buffer Address Attributes** Format: MemoryAddressAttributes
95..96	63:0	**Deblocker Filter Line Read/Write U Buffer Address** Format: SplitBaseAddress64ByteAligned Base address of the filter line buffer (read/write) used by the Deblocking Filter.
97	31:0	**Deblocker Filter Line Read/Write U Buffer Address Attributes** Format: MemoryAddressAttributes
98..99	63:0	**Deblocker Filter Line Read/Write V Buffer Address** Format: SplitBaseAddress64ByteAligned Base address of the filter line buffer (read/write) used by the Deblocking Filter.
100	31:0	**Deblocker Filter Line Read/Write V Buffer Address Attributes** Format: MemoryAddressAttributes
101..102	63:0	**Deblocker Filter Tile Line Read/Write Y Buffer Address** Format: SplitBaseAddress64ByteAligned Base address of the tile line buffer (read/write) used by the Deblocking Filter.
103	31:0	**Deblocker Filter Tile Line Read/Write Y Buffer Address Attributes** Format: MemoryAddressAttributes
104..105	63:0	**Deblocker Filter Tile Line Read/Write V Buffer Address** Format: SplitBaseAddress64ByteAligned Base address of the tile line buffer (read/write) used by the Deblocking Filter.
106	31:0	**Deblocker Filter Tile Line Read/Write V Buffer Address Attributes** Format: MemoryAddressAttributes
107..108	63:0	**Deblocker Filter Tile Line Read/Write U Buffer Address** Format: SplitBaseAddress64ByteAligned Base address of the tile line buffer (read/write) used by the Deblocking Filter.
109	31:0	**Deblocker Filter Tile Line Read/Write U Buffer Address Attributes** Format: MemoryAddressAttributes

AVP_PIPE_BUF_ADDR_STATE		
110..111	63:0	**Deblocker Filter Tile Column Read/Write Y Buffer Address** Format: SplitBaseAddress64ByteAligned Base address of the tile column buffer (read/write) used by the Deblocking Filter.
112	31:0	**Deblocker Filter Tile Column Read/Write Y Buffer Address Attributes** Format: MemoryAddressAttributes
113..114	63:0	**Deblocker Filter Tile Column Read/Write U Buffer Address** Format: SplitBaseAddress64ByteAligned Base address of the tile column buffer (read/write) used by the Deblocking Filter.
115	31:0	**Deblocker Filter Tile Column Read/Write U Buffer Address Attributes** Format: MemoryAddressAttributes
116..117	63:0	**Deblocker Filter Tile Column Read/Write V Buffer Address** Format: SplitBaseAddress64ByteAligned Base address of the tile column buffer (read/write) used by the Deblocking Filter.
118	31:0	**Deblocker Filter Tile Column Read/Write V Buffer Address Attributes** Format: MemoryAddressAttributes
119..120	63:0	**CDEF Filter Line Read/Write Buffer Address** Format: SplitBaseAddress64ByteAligned Base address for the CDEF Filter Line buffer. It includes YUV data
121	31:0	**CDEF Filter Line Read/Write Buffer Address Attributes** Format: MemoryAddressAttributes
122..127	191:0	**Reserved** Access: RO Format: MBZ
128..129	63:0	**CDEF Filter Tile Line Read/Write Buffer Address** Format: SplitBaseAddress64ByteAligned Base address for the CDEF Filter Tile Line buffer. It includes YUV data
130	31:0	**CDEF Filter Tile Line Read/Write Buffer Address Attributes** Format: MemoryAddressAttributes
131..136	191:0	**Reserved** Access: RO Format: MBZ
137..138	63:0	**CDEF Filter Tile Column Read/Write Buffer Address** Format: SplitBaseAddress64ByteAligned Base address for the CDEF Filter Tile Column buffer. It includes YUV data
139	31:0	**CDEF Filter Tile Column Read/Write Buffer Address Attributes** Format: MemoryAddressAttributes

AVP_PIPE_BUF_ADDR_STATE

140..141	63:0	**CDEF Filter Meta Tile Line Read/Write Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base address for the CDEF Filter Meta Tile Line buffer.
142	31:0	**CDEF Filter Meta Tile Line Read/Write Buffer Address Attributes** Format: **MemoryAddressAttributes**
143..144	63:0	**CDEF Filter Meta Tile Column Read/Write Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base address for the CDEF Filter Meta Tile Column buffer.
145	31:0	**CDEF Filter Meta Tile Column Read/Write Buffer Address Attributes** Format: **MemoryAddressAttributes**
146..147	63:0	**CDEF Filter Top-Left Corner Read/Write Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base address for the CDEF Filter Tile Column buffer.
148	31:0	**CDEF Filter Top-Left Corner Read/Write Buffer Address Attributes** Format: **MemoryAddressAttributes**
149..150	63:0	**Super-Res Tile Column Read/Write Y Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base address for the Super-ResolutionTile Column buffer.
151	31:0	**Super-Res Tile Column Read/Write Y Buffer Address Attributes** Format: **MemoryAddressAttributes**
152..153	63:0	**Super-Res Tile Column Read/Write U Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base address for the Super-Resolution Tile Column buffer.
154	31:0	**Super-Res Tile Column Read/Write U Buffer Address Attributes** Format: **MemoryAddressAttributes**
155..156	63:0	**Super-Res Tile Column Read/Write V Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base address for the Super-Resolution Tile Column buffer.
157	31:0	**Super-Res Tile Column Read/Write V Buffer Address Attributes** Format: **MemoryAddressAttributes**
158..159	63:0	**Loop Restoration Filter Tile Column Read/Write Y Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base address for the Loop Restoration Filter Tile Column buffer.
160	31:0	**Loop Restoration Filter Tile Column Read/Write Y Buffer Address Attributes** Format: **MemoryAddressAttributes**
161..162	63:0	**Loop Restoration Filter Tile Column Read/Write U Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base address for the Loop Restoration Filter Tile Column buffer.

AVP_PIPE_BUF_ADDR_STATE

163	31:0	**Loop Restoration Filter Tile Column Read/Write U Buffer Address Attributes** Format: **MemoryAddressAttributes**
164..165	63:0	**Loop Restoration Filter Tile Column Read/Write V Buffer Address** Format: **SplitBaseAddress64ByteAligned** Base address for the Loop Restoration Filter Tile Column buffer.
166	31:0	**Loop Restoration Filter Tile Column Read/Write V Buffer Address Attributes** Format: **MemoryAddressAttributes**
167..169	95:0	**Reserved** Access: RO Format: MBZ
170..175	191:0	**Reserved** Access: RO Format: MBZ
176..177	63:0	**Decoded Frame Status/Error Buffer Base Address** Format: **SplitBaseAddress64ByteAligned** **Decoder Mode :** Specifies the 64 byte aligned buffer address for writing a single status/error cache-line sized record into memory when the Pic Status/Error Report Enable is set in the AVP_PIPE_MODE_SELECT command. The pic status/error record is written by hardware after the picture is decoded.
178	31:0	**Decoded Frame Status/Error Buffer Base Address Attributes** Format: **MemoryAddressAttributes**
179..180	63:0	**Decoded Block Data Streamout Buffer Address** Exists If: //Decoder Only Format: **SplitBaseAddress64ByteAligned** Buffer address for outputting the per-block indirect data to memory when **StreamOutEnable** is set in the AVP_PIPE_MODE_SELECT command. For Decoder: this field is used for transcoding purpose.
181	31:0	**Decoded Block Data Streamout Buffer Address Attributes** Exists If: //Decoder Only Format: **MemoryAddressAttributes**
182..184	95:0	**Reserved**
185..187	95:0	**Reserved**
188..189	63:0	**Original Uncompressed Picture Source Buffer Address** Format: **SplitBaseAddress64ByteAligned** Encoder-Only : Specifies the 64 byte aligned buffer address for the original source pixel frame prior to downscaling

AVP_PIPE_BUF_ADDR_STATE

190	31:0	**Original Uncompressed Picture Source Buffer Address Attributes** Format: MemoryAddressAttributes
191..192	63:0	**Downscaled Uncompressed Picture Source Buffer Address** Format: SplitBaseAddress64ByteAligned Encoder-Only : Specifies the 64 byte aligned buffer address for the downscaled source input pixel frame. This surface pixels are used for encoding frame
193	31:0	**Downscaled Uncompressed Picture Source Buffer Address Attributes** Format: MemoryAddressAttributes
194..195	63:0	**Tile Size Streamout Buffer Address** Format: SplitBaseAddress64ByteAligned Encoder-Only: Specifies the 64 byte aligned buffer address for streaming out TILE size
196	31:0	**Tile Size Streamout Buffer Address Attributes** Format: MemoryAddressAttributes
197..198	63:0	**Tile Statistics Streamout Buffer Address** Format: SplitBaseAddress64ByteAligned Encoder-Only: Specifies 64byte aligned buffer address for streaming out TILE statistic counters including SSE stats
199	31:0	**Tile Statistics Streamout Buffer Address Attributes** Format: MemoryAddressAttributes
200..201	63:0	**CU Streamout Buffer Address** Format: SplitBaseAddress64ByteAligned Encoder-Only: Specifies 64byte aligned buffer address for streaming out CU statistic counters
202	31:0	**CU Streamout Buffer Address Attributes** Format: MemoryAddressAttributes
203..204	63:0	**SSE Line Read/Write Buffer Address** Format: SplitBaseAddress64ByteAligned Description Encoder-Only: Specifies 64byte aligned buffer address for SSE Line Row Store within Tile
205	31:0	**SSE Line Read/Write Buffer Address Attributes** Format: MemoryAddressAttributes
206..207	63:0	**SSE Tile Line Read/Write Buffer Address** Format: SplitBaseAddress64ByteAligned Encoder-Only: Specifies 64byte aligned buffer address for SSE Tile Line Row Store for across tiles
208	31:0	**SSE Tile Line Read/Write Buffer Address Attributes** Format: MemoryAddressAttributes

AVP_PIPE_BUF_ADDR_STATE		
209..210	63:0	**PostCDEF pixels Buffer Address** Format: **SplitBaseAddress64ByteAligned** Encoder-Only: Specifies 64byte aligned buffer address for writing out reconstructed post CDEF filtered pixels
211	31:0	**PostCDEF pixels Buffer Address Attributes** Format: **MemoryAddressAttributes**

AVP_PIPE_MODE_SELECT

AVP_PIPE_MODE_SELECT

Source:	VideoCS
Length Bias:	2

The AVP Pipeline is selected with the **Media Instruction Opcode "8h"** for all AVP Commands. Each AVP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The workload for the AVP pipeline is tile based. Once the bit stream DMA is configured with the AVP_BSD_OBJECT command for a tile in a frame, and the tile's bitstream is presented to the AVP, the tile decoding will begin.
AVP pipeline is stateless, i.e. there is no states saved between the decoding of each tile. Hence all sequence, frame and segment state commands have to be resent before the tile coding command and the BSD object command.
The AVP_PIPE_MODE_SELECT command is responsible for general pipeline level configuration that would normally be set once for a single stream encode or decode and would not be modified on a frame workload basis.
This is a frame level state command and is shared by both encoding and decoding processes.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 3h Codec/Engine Name Format: OpCode Codec/Engine Name = AVP = 3h
	22:16	**Media Instruction Command** Default Value: 0h AVP_PIPE_MODE_SELECT Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Format: =n (Excludes Dwords 0, 1). Value: 5h Name:
1	31:24	**Reserved**

<table>
<tr><th colspan="3">AVP_PIPE_MODE_SELECT</th></tr>
<tr><td></td><td>23</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>22:17</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>16:15</td><td><b>Pipe working Mode</b>
This programs the working mode for AVP pipe.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>00b</td><td>Legacy decoder/encoder mode (Single pipe)</td><td>This is for single pipe mode standalone mode. It is used by both decoder and encoder.</td></tr>
<tr><td>01b</td><td>Reserved</td><td></td></tr>
<tr><td>10b</td><td>Encoder mode (Scalable Multi-pipe)</td><td>This is for multiple-pipe scalable mode for encoder model only. In encoder, it is for PAK.</td></tr>
<tr><td>11b</td><td>Decoder Scalable mode with MSAC in real tiles (Scalable Multi-pipe)</td><td>This is for multiple-pipe scalable mode decoder mode in real tiles. MSAC and reconstruction will run together. Each pipes will run in real tiles vertically.</td></tr>
</table></td></tr>
<tr><td></td><td>14:13</td><td><b>Multi-Engine Mode</b>
This indicates the current pipe is in single pipe mode or if in scalable mode is in left/right/middle pipe in multi-engine mode.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>00b</td><td>Single Engine Mode or MSAC FE only decode mode</td><td>This is for single engine mode (legacy) OR MSAC FE only decode mode
During AV1Decoder Scalability Real Tile Mode, for the last phase, it is possible to have single tile column left. In this case, it should be programmed with pipe as a single engine mode (using this value).
For example, for 9 tile column running on 4 pipes. The first two phases will use all 4 pipes and finish 8 tile column. The remaining one column will be processed as last third phase as single tile column.</td></tr>
<tr><td>01b</td><td>Pipe is the left engine in a Multi-engine mode</td><td>Current pipe is the most left engine while running in scalable multi-engine mode</td></tr>
<tr><td>10b</td><td>Pipe is the right engine in a Multi-engine mode</td><td>Current pipe is the most right engine while running in scalable multi-engine mode</td></tr>
<tr><td>11b</td><td>Pipe is one of the middle engine in a Multi-engine mode</td><td>Current pipe is in one of the middle engine while running in scalable multi-engine mode</td></tr>
</table></td></tr>
<tr><td></td><td>12</td><td><b>Tile Statistics Streamout Enable</b>
Enables Tile level statistics like #of inter/intra PUs, TUsize 4x4,8x8 etc. including SSE per Tile
Encoder Only</td></tr>
</table>

AVP_PIPE_MODE_SELECT

11 **Reserved**

Access:	RO
Format:	MBZ

10 **Reserved**

Access:	RO
Format:	MBZ

9:8 **Reserved**

Access:	RO
Format:	MBZ

7:5 **Codec Standard Select**

Value	Name
2	AV1

4 **Frame reconstruction disable**

Description
Disable writing out reconstructed frames. By default should be 0 Normally writing out B-frames can be disabled for bandwidth/power savings in encoder mode. This bit must be zero in decoder mode

Programming Notes
Even when reconstruction is disabled CDEF output is written out. so, in AVP_PIPE_BUFF_addr command CDEF output surface address should be programmed.

3 **Pic Status/Error Report Enable**

Format:	Enable

Value	Name	Description
0	Disable	Disable status/error reporting
1	Enable	Status/Error reporting is written out once per picture. The Pic Status/Error Report ID in DWord3 along with the status/error status bits are packed into one cache line and written to the Status/Error Buffer address in the AVP_PIPE_BUF_ADDR_STATE command. Must be zero for encoder mode.

2:1 **Reserved**

Format:	MBZ

0 **Codec Select**

Format:	U1

<table>
<tr><th colspan="3">AVP_PIPE_MODE_SELECT</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Decode</td></tr><tr><td>1</td><td>Encode</td></tr></table></td></tr>
<tr><td>2</td><td>31:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>3</td><td>31:0</td><td><b>Pic Status/Error Report ID</b><table><tr><td>Format:</td><td>U32</td></tr></table>The Pic Status/Error Report ID is a unique 32-bit unsigned integer assigned to each picture status/error output. Must be zero for encoder mode.<table><tr><th>Programming Notes</th></tr><tr><td>Software must program different Status/Error Buffer addresses between pictures; otherwise the hardware might overwrite previously written data.</td></tr></table></td></tr>
<tr><td>4</td><td>31:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>5</td><td>31:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>6</td><td>31:7</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>6</td><td><b>Reserved (for SSE Enable in future project)</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Enables SSE metrics calculation and streamout.
Encoder Only</td></tr>
<tr><td></td><td>5</td><td><b>Source Pixel Prefetch Enable</b>
Enables source pixel prefetch.
When set, PAK makes one request for every few SBs(prefetch length) to warm up TLBs before actual requests are made
This bit is used in AV1PAK only mode
Default: Enable
Encoder Only</td></tr>
<tr><td></td><td>4:2</td><td><b>Source Pixel Prefetch Length</b>
This field indicates how often (number of LCUs) PAK should make prefetch request for source pixel.
ValidRange:4-7and mapped as100->2, 101->4, 110->8 and 111->16 LCUs
This field is valid when Source Pixel PreFetch Enabled
Default Value:101 (4 LCUs)
This bit is used for AV1 PAK only Mode</td></tr>
</table>

AVP_PIPE_MODE_SELECT			
	1:0	**Reserved**	
		Access:	RO
		Format:	MBZ

AVP_SEGMENT_STATE

AVP_SEGMENT_STATE

Source:	VideoCS
Length Bias:	2

When segmentation is enabled, this Segment State command is issued once per segment. There can be maximum 8 segments specified to decode a given frame, so this Segment State Command can be issued maximum 8 times. It is assumed that there is no gap in segment IDs. So, when the AVP PIC States specified that the number of active

When segmentation is disabled, driver still sends out this command once for segment id = 0. HW needs to check the segmentation enable flag from AVP_PIC_STATE Command as well to distinguish from the case when segmentation is enabled for segment id = 0.

Each segment can have its own specification of enabling any one of the 8 features defined in AV1 and their corresponding feature data. When a feature is not enabled, its feature data is defaulted to 0. When segmentation is not enabled, all the features are disabled and their corresponding feature data are set to 0.

Segment State Command also provides other segment related parameters.

It is assumed that HW is keeping a copy of the complete AV1 QM Matrix Table for all color components inside its internal memory, and Driver only needs to send the qm_level as index into this Table.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 3h Codec/Engine Name Format: OpCode Codec/Engine Name = AVP = 3h
	22:16	**Media Instruction Command** Default Value: 32h AVP_SEGMENT_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 2h — Name:

AVP_SEGMENT_STATE

1	31:3	**Reserved** Access: RO Format: MBZ
	2:0	**Segment ID** Format: U3 The Segment ID specifies which one of the 8 possible segments that the current Segment State Command is associated with. Segment ID is in the range of [0 ... 7]. Maximum, there can be 8 segments specified for decoding a given frame. Segment ID=0, even when segmentation is disabled.
2	31:28	**Segment Chroma V QM Level** Format: U4 It specifies one of the 16 QM matrices to be used for the Chroma V component of the current segment. It is in the range of [0..15]. QM Level = 15, is a bypass, with no scaling. Default is 15. If the current segment is coded as lossless or if qm matrix is not enabled (frame level SE : using_qmatrix = = 0), then Segment QM Level is set to 15. Chroma V qmlevel = (segment lossless flag || using_qmatrix == 0) ? 15: qm_v (frame level SE). If separate_uv_delta_q (sequence level SE) is set to 0, Segment Chroma V QM Level is set to the same value as of Segment Chroma UQM Level. If separate_uv_delta_q (sequence level SE) is set to 1, Segment Chroma V QM Level can be set independently to a different value as of Segment Chroma U QM Level. For a given frame, all lossy segments should have the same value for Vqm_level.
	27:24	**Segment Chroma U QM Level** Format: U4 It specifies one ofthe 16 QMmatrices to be used for the Chroma U component of the current segment. It is in the range of [0..15]. QM Level = 15, is a bypass, with no scaling. Default is 15. If the current segment is coded as lossless or if qm matrix is not enabled ((frame level SE : using_qmatrix = = 0), then Segment QM Level is set to 15. Chroma U qmlevel = (segment lossless flag || using_qmatrix == 0) ? 15: qm_u (frame level SE). For a given frame, all lossy segments should have the same value for Uqm_level.
	23:20	**Segment Luma Y QM Level** Format: U4 It specifies one of the 16 QMmatrices to be used for the Luma Y component of the current segment. It is in the range of [0..15]. QM Level = 15, is a bypass, with no scaling. Default is 15. If the current segment is coded as lossless or if qm matrix is not enabled (frame level SE : using_qmatrix = = 0), then Segment QM Level is set to 15. Luma qmlevel = (segment lossless flag || using_qmatrix == 0) ? 15: qm_y (frame level SE). For a given frame, all lossy segments should have the same value forY qm_level.
	19	**Segment Lossless Flag** Format: U1

AVP_SEGMENT_STATE

		0 : the current segment is not coded as lossless. Default is 0. 1 : the current segment is coded as lossless. Segment Lossless Flag = (clamp[(base_qindex + optional segment delta qindex), 0, 255] == 0) && (y_dc_delta_q == 0) && (u_dc_delta_q == 0) && (u_ac_delta_q == 0) && (v_dc_delta_q == 0) && (v_ac_delta_q == 0) It is computing usingsyntax elements all from the bitstream, read in the uncompressed header. In encoder mode, if one of the segments is lossless then all segments must be lossless in the frame.
	18	**Segment Block GlobalMV Flag** Format: U1 It specifies the segment feature data corresponding to thefeature id = SEG_LVL_GLOBALMV = 7. Segment Block GlobalMV Flag = 0, specifies nothing. Segment Block GlobalMV Flag = 1, specifies the blocks of the current segment are all coded with Y inter prediction mode set to GLOBALMV. There is no feature data read from the bitstream for setting the Segment Block GlobalMV Flag. It iis derived from it corresponding segment feature mask. Segment Block GlobalMV Flag =seg_feature_mask & (1 « SEG_LVL_GLOBALMV) If Segment Block GlobalMV Flag = 1, The Y inter prediction mode is set to ZeroMV (for using zero MV), and RefFrame[0] (??? elaborate) is set to LAST_FRAME, and RefFrame[1] is set to NONE (-1).
	17	**Segment Block Skip Flag** Format: U1 It specifies the segment feature data corresponding to the feature id = SEG_LVL_SKIP = 6. Segment Block Skip Flag = 0, specifies there is no segment block skip. Segment Block Skip Flag = 1, specifies the blocks of the current segment are all skipped. These blocks are then having all coefficients and MV set to 0 (Y inter prediction mode is set to zero MV mode), but these blocks can still have coding mode specified in the bitstream. There is no feature data read from the bitstream for setting the Segment Block Skip Flag. It iis derived from it corresponding segment feature mask. Segment Block Skip Flag =seg_feature_mask & (1 « SEG_LVL_SKIP) If Segment Block Skip Flag = 1, no coefficient is present in the bitstream, and are all set to 0. The Y inter prediction mode is set to ZeroMV (for using zero MV), and RefFrame[0] is set to LAST_FRAME , and RefFrame[1] is set to NONE (-1). It is used to derived the skip_mode flag and skip flag.
	16:8	**Segment Delta Qindex** Format: S8 It specifies the segment feature data corresponding to the feature id =SEG_LVL_ALT_Q = 0. It is a 9-bit signed 2's complement number, in the range of [-255, +255]. Default is 0. Value -256 is not allowed.
	7:0	**Segment Feature Mask** Format: U8 It is an 8-bit mask specifies which of the 8 segment features are active in the current segment specifications. It is also known as feature enable mask or feature active mask.

AVP_SEGMENT_STATE

		This mask can indicate 0 or up to 8 features are enabled. Any feature can be active or inactive independent of the other features. seg_feature_mask & (1 « feature_id) = 0, the feature (identified by feature_id) is not enabled/active in the current segment. seg_feature_mask & (1 « feature_id) = 1, the feature (identified by feature_id) is enabled/active in the current segment. Feature_ID (enum)Feature Name 0 SEG_LVL_ALT_Q, // Use alternate Quantizer. 1 SEG_LVL_ALT_LF_Y_V, // Use alternate loop filter value on y plane vertical. 2 SEG_LVL_ALT_LF_Y_H, // Use alternate loop filter value on y plane horizontal. 3 SEG_LVL_ALT_LF_U, // Use alternate loop filter value on u plane. 4 SEG_LVL_ALT_LF_V, // Use alternate loop filter value on v plane. 5 SEG_LVL_REF_FRAME, // Optional Segment reference frame. 6 SEG_LVL_SKIP,// Optional Segment zeroMV (0,0) + skip mode. 7 SEG_LVL_GLOBALMV. // Optional Segment zeroMV (0,0). Each segment has its own seg_feature_mask. When a segment is not being used or when segmentation is disabled, the corresponding seg_feature_mask has all 8-bits set to 0. Default all 8-bits are 0.
3	31	**Reserved** Access: RO Format: MBZ
	30:28	**Segment Reference Frame** Format: U3 It specifies the segment feature data corresponding to the feature id =SEG_LVL_REF_FRAME = 5. It is an alternate specification of one of the 8 possible reference frames for a Motion Compensation. =0 for INTRA_FRAME =1 for LAST_FRAME =2 for LAST2_FRAME =3 for LAST3_FRAME =4 for GOLDEN_FRAME =5 for BWDREF_FRAME =6 for ALTREF2_FRAME =7 for ALTREF_FRAME It is in the range of [0..7]. Default is 0.
	27:21	**Segment Delta Loop Filter Level Chroma V** Format: S6 It specifies the segment feature data corresponding to the feature id =SEG_LVL_ALT_LF_V= 4. It is associated with filter_level_v read from the uncompressed header. It is a 7-bit signed 2's complement number, in the range of [-63, +63]. Default is 0.
	20:14	**Segment Delta Loop Filter Level Chroma U** Format: S6 It specifies the segment feature data corresponding to the feature id =SEG_LVL_ALT_LF_U= 3. It is associated with filter_level_u read from the uncompressed header.

AVP_SEGMENT_STATE		
		It is a 7-bit signed 2's complement number, in the range of [-63, +63]. Default is 0.
	13:7	**Segment Delta Loop Filter Level Luma Horizontal** Format: S6 It specifies the segment feature data corresponding to the feature id =SEG_LVL_ALT_LF_Y_H= 2. It is associated with filter_level[1] read from the uncompressed header. It is a 7-bit signed 2's complement number, in the range of [-63, +63]. Default is 0.
	6:0	**Segment Delta Loop Filter Level Luma Vertical** Format: S6 It specifies the segment feature data corresponding to the feature id =SEG_LVL_ALT_LF_Y_V = 1. It is associated with filter_level[0] read from the uncompressed header. It is a 7-bit signed 2's complement number, in the range of [-63, +63]. Default is 0.

AVP_SURFACE_STATE

AVP_SURFACE_STATE

Source:	VideoCS
Length Bias:	2

The AVP Pipeline is selected with the **Media Instruction Opcode "8h"** for all AVP Commands. Each AVP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The AVP_SURFACE_STATE command is responsible for defining the frame buffer pitch and the offset of the chroma component.
This is a picture level state command and is shared by both encoding and decoding processes.
For Decoder, this command is issued once per surface type. There is one reconstructed surface, 8 reference pictures surfaces and one optional IntraBC Decoded Surface (only if IBC is ON).
For Encoder, this command is issued once per surface type. There are4 surface types :source down scaled, source original, reference and reconstructed picture. All reference frames are defined with the same surface command.
Tile-Yf and Tile-Ys are not supported, but HW interface still need to keep these bits as reserved bits.
Note : When NV12 and Tile Y are being used, full pitch and interleaved UV is always in use. U and V X offset must be set to 0; U and V Yoffset must be 8-pixel aligned. For 10-bit pixel, P010 surface definition is being used.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 3h Codec/Engine Name Format: OpCode Codec/Engine Name = AVP = 3h
	22:16	**Media Instruction Command** Default Value: 1h SURFACE_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1).

AVP_SURFACE_STATE

Value	Name
1h	

1 31:28 **Surface Id**

Format:	U4

Value	Name	Description	Programming Notes
0h	Reconstructed Picture	This is for the reconstructed picture surface state	
1h	Source Downscaled Input Picture (encoder only)	Downscaled source pixels used for encoding (creating bitstream) Valid for encoder only	
3h	AV1 Original/Upscaled Source pixels(Encoder Only)	This is for AV1 original/upscaled source pixels surface used for Wiener filter Valid for encoder only	
6h	AV1 INTRA FRAME	This is for AV1 Intra Frame (Reference Picture 0). Each AV1 Reference Pictures can have different size so a separate ID is needed.	
7h	AV1 Last Frame	This is for AV1Last Frame (Reference Picture 1). Each AV1 Reference Pictures can have different size so a separate ID is needed.	
8h	AV1 Last2 Frame	This is for AV1 Last2 Frame (Reference Picture 2). Each AV1 Reference Pictures can have different size so a separate ID is needed.	
9h	AV1 Last3 Frame	This is for AV1 Last3 Frame (Reference Picture 3). Each AV1 Reference Pictures can have different size so a separate ID is needed.	
Ah	AV1 Golden Frame	This is for AV1 Golden Frame (Reference Picture 4). Each AV1 Reference Pictures can have different size so a separate ID is needed.	
Bh	AV1 Bwdref Frame	This is for AV1 Bwdref Frame (Reference Picture 5). Each AV1 Reference Pictures can have different size so a separate ID is needed.	
Ch	AV1 Altref2 Frame	This is for AV1 Altref2 Frame (Reference Picture 6). Each AV1 Reference Pictures can have different size so a separate ID	

AVP_SURFACE_STATE

		is needed.	
Dh	AV1 Altref Frame	This is for AV1 Altref Frame (Reference Picture 7). Each AV1 Reference Pictures can have different size so a separate ID is needed.	
Eh	IntraBC Decoded Frame	This is for AV1 IntraBC Decoded Frame. It will be used both Read/Write at the same time. This surface requires multiple of 8 pixels on both width and height.	This surface must be programmed as uncompressed
Fh	AV1 CDEF pixels streamout(Encoder Only)	Surface for writing out Post CDEF pixels as, For 8bit: NV12 format For 10bit: P010V format WITHOUT LSBs	

27:17 **Reserved**

Access:	RO
Format:	MBZ

16:0 **Surface Pitch Minus1**

Format:	U17-1

This field specifies the surface pitch in (#Bytes - 1).

Programming Notes
For TileYF and TileYS surfaces, the range is dependent on the Cu parameter (refer to Memory Data Formats section for the definition of the Cu parameter depending on the case). The range in bytes is [2^{Cu}-1, 131071] -> [(2^{Cu})B, 128KB] = [1 tile, 128KB/(2$^{Cu) tiles}$] The field specifies the surface pitch in (#Bytes - 1)
For tiled surfaces, the pitch must be a multiple of the tile width (i.e.128 bytes aligned). If Half Pitch for Chroma is set, this field must be a multiple of two tile widths for tiled surfaces, or a multiple of 2 bytes for linear surfaces. For Y-tiled surfaces: Range = [127, 131071] to [128B,128KB] = [1 tile, 1024 tiles]
For Variant format with packed LSB ("Variant Format LSB Packed Enable" is set), the pitch can be smaller than non-packed format. "Zeros" are inserted between LSB (each pixel LSB is byte-aligned by inserting zero on the lower part of the byte). The number of "zeros" depends on the bitdepth of the pixels (10 bit will have 2 bit in LSB and 6 bits of zeros are added between pixel in LSB part). MinPitch = MSB_Region_in_Bytes + LSB_Region_in_Bytes MSB_Region_in_Bytes= ((PictureWidth in Pixels * 1 Byte) + 31 Bytes)/ 32 [LSBRegion starts on multiple of 32 bytes] LSB_Region_in_Bytes = (((Picture Width in Pixels * (Pixel_Bitdepth_in_Bit - 8 bits) + 7 bits) / 8 bits_per_byte) + 31 Bytes) / 32 [32 bytes alignment after Byte aligned]. For tiled surfaces, the pitch needs to be padded to be multiple of tile width.

<table>
<tr><th colspan="3">AVP_SURFACE_STATE</th></tr>
<tr><td>2</td><td>31:27</td><td><b>Surface Format</b>
<table><tr><td>Format:</td><td>U5</td></tr></table>
Specifies the format of the surface.
P010V is only applied to the Surface Id=Fh (AV1 CDEF pixels streamout), encoder only. And also the 2 LSBs are removed, so this P010V surface is actually an 8-bit surface.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Programming Notes</th></tr>
<tr><td>3h</td><td>P010Variant</td><td>P010Variant is a modified P010 format, >8 bit planar 420 with MSB together and LSB at an offset in x direction where the x-offset should be 32-bit aligned.</td><td>Encoder Only</td></tr>
<tr><td>4h</td><td>PLANAR_420_8</td><td></td><td></td></tr>
<tr><td>Dh</td><td>P010</td><td></td><td></td></tr>
</table></td></tr>
<tr><td></td><td>26</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>25</td><td><b>Variant Format LSB Packed Enable</b>
This bit indicates if the LSB portion of the variant format is packed together or byte-aligned with 0 to lower portion part of the byte.
This is only valid forP010Variant/P016Variant andY210Variant/Y216Variant (444 Variant is not supported currently). This bit must be programmed to 0 for all other format.
<table>
<tr><th>Value</th><th>Name</th><th>Programming Notes</th></tr>
<tr><td>0</td><td>LSB Unpacked</td><td>Indicates LSB portion of the Variant format is byte-aligned per pixel by adding "0" to the lower part of the byte</td></tr>
<tr><td>1</td><td>LSB Packed</td><td>Indicates LSB portion of the Variant format is packed together (multiple LSB pixels are packed together to form a byte). The number of LSB pixels can be packed together depends on the bitdepth of the pixels.
Not supported in Encoder Mode</td></tr>
</table></td></tr>
<tr><td></td><td>24:15</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:0</td><td><b>Y Offset for U(Cb) in pixel</b>
<table><tr><td>Format:</td><td>U15</td></tr></table>
This field specifies the vertical offset in rows from the <b>Surface Base Address</b> to the start(origin) of the U(Cb) plane or the interleaved UV plane if <b>Interleave Chroma</b> is enabled. This field is only used for PLANAR surface formats.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td><ul><li>For PLANAR_420 surface formats, the alignment of this field follows the tile mode described in bits 14:13 of the <b>Memory Address Attributes</b> table.</li><li>TileY (legacy 4k) - 8 pixel aligned</li><li>TileYF (New 4k) - 64 pixel aligned</li></ul></td></tr>
</table></td></tr>
</table>

AVP_SURFACE_STATE

		• TileYS (64k) - 256 pixel aligned
3	31:16	**Y Offset for V(Cr)** Format: U16 Row Offset in Pixels This field specifies the vertical offset in rows from the Surface Base Address to the start (origin) of the V(Cr) plane. This field is only used for PLANAR surface formats with Interleave Chroma disabled. **Programming Notes** • TileY (legacy 4k) - 8 pixel aligned • TileYF (New 4k) - 64 pixel aligned • TileYS (64k) - 256 pixel aligned
	15:0	**Default Alpha Value**
4	31:21	**Reserved** Access: RO Format: MBZ
	20:16	**Compression Format** Format: **Media Compression Format** Specifies the Compression Format.
	15	**Compression Type for Altref Frame** This bit is for AV1 Altref Frame (Reference Picture 7). Valid only when Memory Compression for Altref Frame is enabled. **Value** / **Name** 0 / Media compression Enabled **[Default]** 1 / Render compression Enabled
	14	**Compression Type for Altref2 Frame** This bit is for AV1 Altref2 Frame (Reference Picture 6). Valid only when Memory Compression for Altref2 Frame is enabled. **Value** / **Name** 0 / Media compression Enabled **[Default]** 1 / Render Compression Enabled
	13	**Compression Type for Bwdref Frame** This bit is for AV1 Bwdref Frame (Reference Picture 5). Valid only when Memory Compression for Bwdref Frame is enabled. **Value** / **Name** 0 / Media compression Enabled **[Default]** 1 / Render Compression Enabled

<table>
<tr><th colspan="3">AVP_SURFACE_STATE</th></tr>
<tr><td></td><td>12</td><td><b>Compression Type for Golden Frame</b>
This bit is for AV1 Golden Frame (Reference Picture 4). Valid only when Memory Compression for Golden Frame is enabled.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>11</td><td><b>Compression Type for Last3 Frame</b>
This bit is for AV1 Last3 Frame (Reference Picture 3). Valid only when Memory Compression for Last3 Frame is enabled.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Compression Type for Last2 Frame</b>
This bit is for AV1 Last2 Frame (Reference Picture 2). Valid only when Memory Compression for Last2 Frame is enabled.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>Compression Type for Last Frame</b>
This bit is for AV1 Last Frame (Reference Picture 1). Valid only when Memory Compression for Last Frame is enabled.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>8</td><td><b>Compression Type for Intra Frame</b>
This bit is for Intra Frame (Reference Picture 0). Valid only when Memory Compression for Intra Frame is enabled.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>7</td><td><b>Memory Compression Enable for AV1 Altref Frame</b>
This bit is for AV1Altref Frame (Reference Picture 7).<table><tr><th>Value</th><th>Name</th></tr><tr><td>1</td><td>Memory Compression Enable</td></tr><tr><td>0</td><td>Memory Compression Disable</td></tr></table></td></tr>
<tr><td></td><td>6</td><td><b>Memory Compression Enable for AV1 Altref2 Frame</b>
This bit is for AV1Altref2 Frame (Reference Picture 6).<table><tr><th>Value</th><th>Name</th></tr><tr><td>1</td><td>Memory Compression Enable</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">AVP_SURFACE_STATE</th></tr>
<tr><td></td><td></td><td><table><tr><td>0</td><td>Memory Compression Disable</td></tr></table></td></tr>
<tr><td></td><td>5</td><td><b>Memory Compression Enable for AV1 Bwdref Frame</b>
This bit is for AV1 Bwdref Frame (Reference Picture 5).<table><tr><th>Value</th><th>Name</th></tr><tr><td>1</td><td>Memory Compression Enable</td></tr><tr><td>0</td><td>Memory Compression Disable</td></tr></table></td></tr>
<tr><td></td><td>4</td><td><b>Memory Compression Enable for AV1 Golden Frame</b>
This bit is for AV1 Golden Frame (Reference Picture 4).<table><tr><th>Value</th><th>Name</th></tr><tr><td>1</td><td>Memory Compression Enable</td></tr><tr><td>0</td><td>Memory Compression Disable</td></tr></table></td></tr>
<tr><td></td><td>3</td><td><b>Memory Compression Enable for AV1 Last3 Frame</b>
This bit is for AV1 Last3 Frame (Reference Picture 3).<table><tr><th>Value</th><th>Name</th></tr><tr><td>1</td><td>Memory Compression Enable</td></tr><tr><td>0</td><td>Memory Compression Disable</td></tr></table></td></tr>
<tr><td></td><td>2</td><td><b>Memory Compression Enable for AV1 Last2 Frame</b>
This bit is for AV1 Last2 Frame (Reference Picture 2).<table><tr><th>Value</th><th>Name</th></tr><tr><td>1</td><td>Memory Compression Enable</td></tr><tr><td>0</td><td>Memory Compression Disable</td></tr></table></td></tr>
<tr><td></td><td>1</td><td><b>Memory Compression Enable for AV1 Last Frame</b>
This bit is for AV1 Last Frame (Reference Picture 1).<table><tr><th>Value</th><th>Name</th></tr><tr><td>1</td><td>Memory Compression Enable</td></tr><tr><td>0</td><td>Memory Compression Disable</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>Memory Compression Enable for AV1 Intra Frame</b>
This bit is for AV1 Intra Frame (Reference Picture 0).<table><tr><th>Value</th><th>Name</th></tr><tr><td>1</td><td>Memory Compression Enable</td></tr><tr><td>0</td><td>Memory Compression Disable</td></tr></table></td></tr>
</table>

AVP_TILE_CODING

<table>
<tr><th colspan="3">AVP_TILE_CODING</th></tr>
<tr><td colspan="3">Source: BSpec
Length Bias: 1</td></tr>
<tr><td colspan="3"><table><tr><th>Programming Notes</th></tr><tr><td>This command is used only for AV1codec. It is issued for every tile of a frame. If a frame is composed of only 1 tile, it is still being issued. Tiling and Tile Group organization in AV1 cannot be disabled, a frame minimum must have 1 tile. Currently, each batch buffer can contain only 1 tile to be processed, it cannot contain more than 1 tile or the entire tile group of tiles.
When the tile width exceeds 4096 pixels or the tile area exceeds 4096x2304 pixels, tiling must be performed and number of tiles in such frame must be >1. There is no mandatory tiling driven by tile height. The frame height in pixels will limit the allowed tile height in extreme situation. Hence, the AVP_TILE_CODING can be issued multiple times for decoding a frame.
Since AVP HW pipeline is stateless, all sequence, frame and segment level states (coding parameters in all Frame Level State Commands) must be resent before sending each TILE_CODING_STATE command.
Although tile size is specified in SuperBlock unit, the minimum tile size is actually set to be 8x8 pixels (which is the same as the minimum frame size in pixels). It can also happen to the rightmost tile column and bottommost tile row of a frame which is not divisible by the SuperBlock size - this leads to the presence of partial tile and partial SuperBlock handling.
AV1 supports both
1) a uniform-spacing tiling scheme (as in VP9, which is always in the form of 2^N x 2^M number of tiles, for the entire frame), and
2) a non-uniform-spacing tiling scheme. Bitstream syntax elements will specify the width and height of each tile size in the frame.
AVP HW pipeline is a tile-based codec engine, it does not need to distinguish between these two tiling schemes. Driver will take care of the difference and details of these tiling schemes. At the end, Driver will send out one tile at a time with all the related tile information to the HW through this TILE_CODING State Command.
In AV1, a frame is partitioned by tile row and tile column. That is, a tile boundary must go across the full frame width or the full frame height only. There is no tiling within a tile.
For AV1, the max number of tiles per frame is set to 256 in the LEVEL definition for regular video decoding. The ext-tile (Virtual Reality mode, currently not supported) has a different tiling configuration, constraints and definition.</td></tr></table></td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline Type</b><table><tr><td>Default Value:</td><td>2h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>26:23</td><td><b>Media Instruction Opcode</b><table><tr><td>Default Value:</td><td>3h Codec/Engine Name</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

AVP_TILE_CODING

	22:16	**Media Instruction Command** Default Value: 15h AVP_TILE_CODING Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n Excludes Dwords 0 & 1 **Value** — **Name** 4h — **[Default]** 5h — **[Default]**
1	31:24	**Tile Group ID** Format: U8 **Description** It specifies the Tile Group the current tile belongs to. It is intel specific parameter, not present in the AV1 bitstream. Each tile is belonging to a Tile Group, and there can be multiple TGs in a frame. All TGs of a frame are lined up in raster order within the frame. Each TG is received from the bitstream as a separate TG_OBU. So, a frame can receive multiple TG_OBUs. Currently, all TG_OBUs of a frame must be received in the correct sequential order (not arbitrary order is defined yet). A TG can start at any tile position of a frame (e.g., at the beginning of a tile row, in the middle of a tile row, or at the end of a tile row). A TG does not break by the frame width, and can continue to the next row of tiles below. A TG can span multiple tile rows. A TG can end at any tile position of a frame(e.g. at the beginning of a tile row, in the middle of a tile row, or at the end of a tile row). A TG can be viewed as a linear chain of tiles (not necessary a 2-D rectangular/square region of a frame). A TG can maximum contain the whole frame of tiles, or minimum contain only a single tile of the frame. A frame can be entirely coded as a single tile in a single TG. Each TG is assigned a unique ID number, in raster increasing order. This numbering is intel specific, not defined in the spec. There is no jump or gap in TG ID value between two adjacent TG. The top-left most tile of a frame is always TG ID = 0. The bottom-right most tile of a frame is always the highest TG ID value. Tile Group ID is in the range [0..255]. The max value 255 is intel specific. Note: the max number of TG per frame is set equal to the max number of tiles allowed in a frame, since each tile can belong to a different TG. Encoder: Only single tile group supported
	23:12	**TG Tile Num** Format: U12

AVP_TILE_CODING

		Description Specify the Tile Number inside a Tile Group. This numbering is intel specific, not present in the AV1 bitstream. All tiles of a TG are numbered in sequential raster order, starting from 0. TG Tile Num is reset at each TG boundary. That is, the very first tile of a TG is always assigned a TG Tile Num of 0. The last tile of a TG has the largest Tile Num of that TG. So, the total number of tiles in a frame = (the largest Frame Tile ID + 1) = sum for all TG [largest TG Tile Num+ 1]. Max number of tiles in a TG is currently limited to 128 at the highest level 6.3 being defined for regular video. TG Tile Num is in the range [0..255], starting from 0. The upper 5bits are reserved for future expansion. Encoder: Supports TG Tile Num in the range [0..254]
	11:0	**Frame Tile ID** Format: U12 All the tiles of a frame are labeled with a Frame Tile ID in a sequence raster order, starting from 0. The very first tile of a frame (top-left most corner) has Frame Tile ID = 0, and the last tile of a frame (bottom-right most corner)has the largest Frame Tile ID. So, the total number of tiles in a frame = (the largest Frame Tile ID + 1) = sum for all TG [largest Tile Num in a TG + 1]. Frame Tile ID is numbered from the 2 TG syntax elements TG_START and TG_END, or their default value when no present in the bitstream. Max number of tiles per frame is currently limited to 128 at the highest level 6.3 being defined for regular video. But for VR Large Scale Tile, max number of tiles per tile list is 512.Frame Tile ID is in the range [0..511], starting from 0.The upper 3bits are reserved for future expansion. Frame Tile ID numbering is consistent with the spec definition of tile ordering in a Tile Group and in a frame. Frame Tile ID does not get reset to 0 at Tile Group boundary (as the TG Tile Num does). Frame Tile ID numbering is not affected by dependency setting across tiles.
2	31:26	**Reserved** Access: RO Format: MBZ
	25:16	**Tile Row Position in SB Unit** Format: U10 Specify the row (y-) position of the current tile to be processed in a frame. The position is specified in SuperBlock unit. The SuperBlock unit is specified in the Sequence Header; it can be either 64x64 pixels or 128x128 pixels. For regular video, the decoded tile pixels are placed in the same tile location in the decoded output frame as the coded tile in the bitstream. But for VR Large Scale Tile, the decoded tile pixels can be placed in a different location predefined in the decoded pseudo output frame as the coded tile in the bitstream. This field is used for both regular video and VR Large Scale Tile. For VR Large Scale Tile, this field is programmed to be the coded tile position for MotionCompensation purpose, and the Output Decoded Tile Row Position field is then programmed to be the position in the decoded pseudo output frame for placing the decoded pixels of the tile,

AVP_TILE_CODING

A Tile contains a 2D array of SB units. A Tile min. size is 1 SB unit. The Tile position is based on the top-left most corner SB unit of the Tile.
The very first tile of a frame (top-left most corner) has the position [Tile Column Position in SB Unit , Tile Row Position in SB Unit] = [0, 0]
The frame height is rounded up to an integer multiple of the Superblock unit.
A frame height that is less than a SB unit (but must be >=8 pixels), is rounded up to 1 SB unit for the purpose of defining a tile.
A frame height that is not divisible by tile height, the last tile row of the frame will have a smaller tile height, but still an integer multiple of SB unit. It does not affect the Tile Row Position in SB Unit of these partial Tiles.
Tile Row Position in SB unit is derived from the frame level syntax elements in tiling specifications (tile_info()).
Max Frame Height = 64K, hence the max number of SB unit= 1024, when the SB unit is 64x64 pixels, and the max Tile Row Position in SB unit can be 1023.
Intel supports Tile Row Position in SB unit only in the range of [0..255], for 16K Max Frame Height. starting from 0 at the top-left most corner of a frame.

Value	Name	Description
[0,255]	16K_Below	This allows support up to 16K picture

15:10 **Reserved**

Access:	RO
Format:	MBZ

9:0 **Tile Column Position in SB Unit**

Format:	U10

Specify the column (x-) position of the current tile to be processed in a frame. The position is specified in SuperBlock unit. The SuperBlock unit is specified in the Sequence Header; it can be either 64x64 pixels or 128x128 pixels.
For regular video, the decoded tile pixels are placed in the same tile location in the decoded output frame as the coded tile in the bitstream. But for VR Large Scale Tile, the decoded tile pixels can be placed in a different location predefined in the decoded pseudo output frame as the coded tile in the bitstream. This field is used for both regular video and VR Large Scale Tile.
For VR Large Scale Tile, this field is programmed to be the coded tile position for MotionCompensation purpose, and the Output Decoded Tile Column Position field is then programmed to be the position in the decoded pseudo output frame for placing the decoded pixels of the tile,
A Tile contains a 2D array of SB units. A Tile min. size is 1 SB unit. The Tile position is based on the top-left most corner SB unit of the Tile.
The very first tile of a frame (top-left most corner) has the position [Tile Column Position in SB Unit , Tile Row Position in SB Unit] = [0, 0]
The frame width is rounded up to an integer multiple of the Superblock unit.
A frame width that is less than a SB unit (but must be >= 8 pixels), is rounded up to 1 SB unit for the purpose of defining a tile.
A frame width that is not divisible by tile width, the last tile column of the frame will have a smaller tile width, but still an integer multiple of SB unit. It does not affect the Tile Column Position in SB Unit of these partial Tiles.

AVP_TILE_CODING

		Tile Column Position in SB unit is derived from the frame level syntax elements in tiling specifications (tile_info()). Max Frame Width= 64K, hence the max number of SB unit= 1024, when the SB unit is 64x64 pixels, and the max Column Row Position in SB unit can be 1023. Intel supports Tile Column Position in SB unit only in the range of [0..255], for 16K Max Frame Width. starting from 0 at the top-left most corner of a frame. <table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>[0,255]</td><td>16K_Below</td><td>This allows support up to 16K picture.</td></tr></table>
3	31:26	**Reserved** <table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table>
	25:16	**Tile Height in SuperBlock Unit Minus1** <table><tr><td>Format:</td><td>U10</td></tr></table> <table><tr><th>Description</th></tr><tr><td>Tile Size is measured in SuperBlock Unit. The SuperBlock unit is specified in the Sequence Header; it can be either 64x64 pixels or 128x128 pixels. When large-scale tile is ON, tile height must be 1 SB in size (i.e. can be 64x64 or 128x128, depending on the SB size flag). The minimum Tile Size is 1 SB unit x 1 SB unit The frame height is rounded up to an integer multiple of the Superblock unit. If a frame height is not divisible by the SuperBlock unit, the bottommost row of SBs of the frame is partial in size, but for the purpose of tile size definition, the partial SB is still counted as 1 unit. A frame height that is less than a SB unit, is rounded up to 1 SB unit for the purpose of defining a tile. A frame height that is not divisible by tile height, the last tile row of the frame will have a smaller tile height, but still an integer multiple of SB unit. Tile Height in SB unit is derived from the frame level syntax elements in tiling specifications (tile_info()). In AV1, there are two max tile size constraints : Max Tile Width <= 4096 pixels and Max Tile Area <= 4096 width x2304 height pixels. But there is no separate constraint for Max Tile Height. Intel set a limit for frame height to be 16K pixels. In AV1, the following restrictions apply: 1) Last SB (if partial in size)at frames bottommost edge must align to 8x8 block (partial SB)</td></tr><tr><td>In AV1, the following additional restrictions apply: 1) In Scalability Mode, the minimum tile size is 2 width x1 height SBs. (Intel restriction)</td></tr></table> <table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>[0,135]</td><td>8704_and_Below</td><td>This supports upto 8704 (in pixels) for level 6.3.</td></tr></table>
	15:6	**Reserved** <table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table>

AVP_TILE_CODING		
	5:0	**Tile Width in SuperBlock Unit Minus1** Format: U6 **Description** Tile Size is measured in SuperBlock Unit. The SuperBlock unit is specified in the Sequence Header; it can be either 64x64 pixels or 128x128 pixels. The minimum Tile Size is 1 SB unit x 1 SB unit The frame width is rounded up to an integer multiple of the Superblock unit. If a width height is not divisible by the SuperBlock unit, the rightmost column of SBs of the frame is partial in size, but for the purpose of tile size definition, the partial SB is still counted as 1 unit. A frame width that is less than a SB unit, is rounded up to 1 SB unit for the purpose of defining a tile. A frame width that is not divisible by tile width, the last tile column of the frame will have a smaller tile width, but still an integer multiple of SB unit. Tile Width in SB unit is derived from the frame level syntax elements in tiling specifications (tile_info()). In AV1, there are two max tile size constraints : Max Tile Width <= 4096 pixels and Max Tile Area <= 4096 width x2304 height pixels. But there is no separate constraint for Max Tile Height. When super-res is ON, these tile constraints are applied to the downscaled frame's tiles. So, Tile Width in SuperBlock Unit Minus1 is in the range of [0..63]. Tile Width = (Tile Width in SuperBlock Unit Minus1 + 1) SBs. In AV1, the following restrictions apply. 1) Last SB (if partial in size)at frames rightmost edge must align to 8x8 block (partial SB) In AV1, the following additional restrictions apply: 1) In Scalability Mode, the minimum tile size is 2 width x1 height SBs. (Intel restriction)
4	31	**Disable Frame Context Update Flag** Format: U1 **Description** Set to 0, the current tile being decoded is writing out its updated Frame Context to memory (surface buffer), at the end of its decoding. Set to 1, the current tile being decoded is not writing out its updated Frame Context to memory, for use in the next frame to be decoded. It is the frame level syntax element, disable_frame_end_update_cdf, or named as [! refresh_frame_context]. Default is 0. The arithmetic decoding of each tile of a frame is started with the same Frame Context (CDF Cumulative Probability Table Set for all AV1 Syntax Elements) provided by the Driver. This is from a read-only memory surface. After each arithmetic decoding of a Syntax Element from the bitstream,its corresponding CDF Table will be updated (if Disable CDF Update Flag = 0). As such, the frame context is changed after decoding each tile. When Backward Adaptation of the current frame context (CDF Cumulative Probability Table Set for all AV1 Syntax Elements) is enabled, only the largest tile in byte size (bitstream size) of the current frame being decoded will update the frame context for the decoding of the next frame. Driver will determine which tile in the current frame will need to write out its updated frame

AVP_TILE_CODING

		context to memory by setting Disable Frame Context Update Flag to 0. This is to a write-only memory surface. For all other tiles of the frame, Driver will set Disable Frame Context Update Flag to 1. If Driver knows, ahead of time, which tile has the largest bitstream size, then the frame context update can only be done once. If Driver does not know, ahead of time, which tile has the largest bitstream size, then the frame context update may be done more than once, and each time overwritten the previous one, until the largest tile has found. Note : Even when Error Resilient Mode is ON, this field can still be 0 or 1. When in Intel Scalability mode (multiple HW pipes), there are still be one read-only and one write-only frame context buffers allocated for decoding the current frame. And at any one time, Driver will only enable one pipe to update/write out the frame context. Again, it is possible that this update can be done more than once until the largest tile among all pipes has found.
	30	**Disable CDF Update Flag** Format: U1 Set to 1, the current tile being decoded is not updating the CDF table of each syntax element after parsing (multisymbol arithmetic decode) from the bitstream. Hence, the frame context is not being changed. And no need to write out the frame context at the end of decoding the frame. Set to 0, the current tile being decoded is updating the CDF table of each syntax element after parsing (multisymbol arithmetic decode) from the bitstream. Hence. the frame context is changed. If Disable Frame Context Update Flag = 0 for this tile, the new frame context is writing out to memory for subsequent frame decoding. It is the frame level syntax element, disable_cdf_update. Default is 0.
	29	**IsLastTileOfFrame Flag** Format: U1 Indicates if current tile being decoded is the last tile of a frame. Default is 0. The last tile is the bottom-right most corner region of a frame. This is intel specific frame level parameter.
	28	**IsEndTileOfTileGroup Flag** Format: U1 Indicates if current tile is the last tile of a Tile Group. Default is 0. 0 - is not the end tile of a tile group 1 - is the end tile of a tile group. It is derived as: isEndTileOfTileGroup = (Frame Tile ID = = TG_END syntax element) A Tile Group can end at any tile in a tile row (e.g. the first tile of a tile row, the last tile of a tile row, or a tile anywhere in the middle of a tile row). The End tile of a Tile Group may not be in the same tile row as the Start tile of the same Tile Group, but must come after in raster order. This is intel specific frame level parameter.
	27	**IsStartTileOfTileGroup Flag** Format: U1 Indicates if current tile is the first tile of a Tile Group. Default is 0.

AVP_TILE_CODING

		0 - is not the start tile of a tile group 1 - is the start tile of a tile group. It is derived as : isStartTileOfTileGroup = (Frame Tile ID = = TG_START syntax element) A Tile Group can start at any tile in a tile row (e.g. the first tile of a tile row, the last tile of a tile row, or a tile anywhere in the middle of a tile row). The End tile of a Tile Group may not be in the same tile row as the Start tile of the same Tile Group. This is intel specific frame level parameter.
	26	**IsLastTileOfRow Flag** Format: | U1 Indicates if the current tile is the last tile of the current tile row. Default is 0. 0 - is not the last tile of a tile row 1 - is the last tile of a tile row. It is derived from Frame Tile ID and Num of Tile Columns Minus1. It is the tile at the right frame boundary. The bottom-right most tile of a frame is having both IsLastTileOfRow and IsLastTileOfColumn set to 1. This is intel specific frame level parameter.
	25	**IsLastTileOfColumn Flag** Format: | U1 Indicates if the current tile is the last tile of the current tile column. Default is 0. 0 - is not the last tile of a tile column 1 - is the last tile of a tile column. It is derived from Frame Tile ID, Num of Tile Columns Minus1and Num of Tile Rows Minus1. is the tile at the bottom frame boundary. The bottom-right most tile of a frame is having both IsLastTileOfRow and IsLastTileOfColumn set to 1. This is intel specific frame level parameter.
	24	**Reserved** Access: | RO Format: | MBZ
	23	**First Tile in a Frame** Indicates First Tile of a Frame for HW to insert header Encoder Only
	22:4	**Reserved** Access: | RO Format: | MBZ
	3:0	**Reserved** Access: | RO Format: | MBZ
5	31:22	**Num of Tile Rows Minus1 in a Frame** Format: | U10

AVP_TILE_CODING

		Specify the total number of Tile Rows in a frame. TileRows =Num of Tile Rows Minus1 in a Frame + 1. Max Frame Height is 64K, the smallest tile size is 1SB unit, hence max number of tile rows in a frame can be 1024. Valid Num of Tile Rows Minus1 in a Frame is in the range of [0..63] only. The max tile number in a frame is governed by the Level definition for a compliant bitstream. For regular video, highest Level defined is 6.3 (max. 8K video), which specifies max total 128 tiles in a frame and max number of tile columns is 16. The corresponding tile configuration is flexible, it can be 16x8, 8x16, 16x4, 4x16, 8x8, etc. There is no constraints on max number of tile rows. For VR large scale tile application, the tile configuration can be max total 64x64=4K tiles in a pseudo output frame and an anchor frame. But the tile list can max. contain 512 tiles at a time. This is the same as the variable tile_rows (minus1)defined in the reference C model. Num of Tile Rows in a frame is derived from the frame level syntax elements in tiling specifications (tile_info()).
	21:12	**Num of Tile Columns Minus1 in a Frame** Format: U10 **Description** Specify the total number of Tile Columns in a frame. TileColumns=Num of Tile Columns Minus1 in a Frame + 1. Max Frame Width is 64K, the smallest tile size is 1SB unit, hence max number of tile columns in a frame can be 1024. Valid Num of Tile Columns Minus1 in a Frame is in the range of [0..63] only. The max tile number in a frame is governed by the Level definition for a compliant bitstream. For regular video, highest Level defined is 6.3 (max. 8K video), which specifies max total 128 tiles in a frame and max number of tile columns is 16. The corresponding tile configuration is flexible, it can be 16x8, 8x16, 16x4, 4x16, 8x8, etc. There is no constraints on max number of tile rows. For VR large scale tile application, the tile configuration can be max total 64x64=4K tiles in a pseudo output frame and an anchor frame. But the tile list can max. contain 512 tiles at a time. This is the same as the variable tile_cols (minus1) defined in the reference C model. Num of Tile Columns in a frame is derived from the frame level syntax elements in tiling specifications (tile_info()). In Intel Scalability Mode (multiple HW pipes), execution across multiple pipes is done in tile column fashion. When the number of tile columns in a frame > the number of HW pipes, the tile columns are cycled through the pipes in multiple phases as described below: Tile Col0/Row0 | Tile Col1/Row0 | Tile Col2/Row0 Tile Col0/Row1 | Tile Col1/Row1 | Tile Col2/Row1 Tile Col0/Row2 | Tile Col1/Row2 | Tile Col2/Row2 Assume there is only 2 HW pipes (Pipe0 and Pipe1) Phase 0: Pipe0 :Tile Col0/Row0 ; Pipe1 : Tile Col1/Row0 Phase 1: Pipe0 :Tile Col2/Row0 Phase 2:

AVP_TILE_CODING

<table>
<tr><td></td><td></td><td><table><tr><td>Pipe0 :Tile Col0/Row1 ; Pipe1 : Tile Col1/Row1
Phase 3:
Pipe0 :Tile Col2/Row1
Phase 4:
Pipe0 :Tile Col0/Row2 ; Pipe1 : Tile Col1/Row2
Phase 5:
Pipe0 :Tile Col2/Row2.</td></tr></table></td></tr>
<tr><td></td><td>11:8</td><td><b>Reserved MBZ</b><table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>Number of Active BE Pipes</b><table><tr><td>Indicates the number of active, consecutive positioned Scalable VDBOXs to be used for the current frame decoding or encoding.
BE Pipe partitioning, SW must guarantee the minimum width is at least two full SBs for each tiles</td></tr><tr><td>This field in general should be smaller or equal to Num of Tile columns in a Frame.
This field is ignored by HW</td></tr><tr><td>This field is not used by HW</td></tr><tr><td><table><tr><th>Value</th><th>Comment</th></tr><tr><td>0</td><td>ignored</td></tr><tr><td>1</td><td>ignored</td></tr><tr><td>2</td><td>Supported by Encoder / Decoder.</td></tr><tr><td>3</td><td>Supported only by Decoder.</td></tr><tr><td>4</td><td>Supported only by Encoder</td></tr></table></td></tr></table></td></tr>
<tr><td>6</td><td>31:26</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>25:16</td><td><b>Output Decoded Tile Row Position in SB Unit</b><table><tr><td>Format:</td><td>U10</td></tr></table>Specify the row (y-) position of the current decoded tile position in a decode pseudo output frame. The position is specified in SuperBlock unit. The SuperBlock unit is specified in the Sequence Header; it can be either 64x64 pixels or 128x128 pixels.
This field is only used in VR Large Scale Tile application. For VR Large Scale Tile, the decoded tile pixels can be placed in a different location predefined in the decoded pseudo output frame as the coded tile in the bitstream. For VR Large Scale Tile, the Tile Row Positionfield is programmed to be the coded tile position for MotionCompensation purpose, and the Output Decoded Tile Row Position field is then programmed to be the position in the decoded pseudo output frame for placing the decoded pixels of the tile.
This field is used, only when Large Scale Tile Enable flag is set to 1; otherwise, HW can ignore this field.</td></tr>
</table>

AVP_TILE_CODING

Value	Name	Description
[0,255]	16K_Below	This allows support up to 16K picture

15:10 **Reserved**

Access:	RO
Format:	MBZ

9:0 **Output Decoded Tile Column Position in SB Unit**

Format:	U10

Specify the row (x-) position of the current decoded tile position in a decode pseudo output frame. The position is specified in SuperBlock unit. The SuperBlock unit is specified in the Sequence Header; it can be either 64x64 pixels or 128x128 pixels.

This field is only used in VR Large Scale Tile application. For VR Large Scale Tile, the decoded tile pixels can be placed in a different location predefined in the decoded pseudo output frame as the coded tile in the bitstream. For VR Large Scale Tile, the Tile Column Positionfield is programmed to be the coded tile position for MotionCompensation purpose, and the Output Decoded Tile Column Position field is then programmed to be the position in the decoded pseudo output frame for placing the decoded pixels of the tile.

This field is used, only when Large Scale Tile Enable flag is set to 1; otherwise, HW can ignore this field.

Value	Name	Description
[0,255]	16K_Below	This allows support up to 16K picture.

AVP_VD_CONTROL_STATE

AVP_VD_CONTROL_STATE

Source:	VideoCS
Length Bias:	2

For AVP, it is selected with the **Media Instruction Opcode "3h".**

This command is used to modify the control of HCP pipe. It can be inserted anywhere within a frame. It can be inserted multiple times within a frame as well.

This command is also used for AVP pipe.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 3h Codec/Engine Name for AVP Format: OpCode Codec/EngineName = AVP = 3h
	22:16	**Media Instruction Command** Default Value: Ah VD_CONTROL_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value | Name 2h |
1..2	63:0	**VD Control State Body** Format: VD_CONTROL_STATE_BODY

Barrier

MSD_BARRIER - Barrier

Source:	EuSubFunctionGateway
Length Bias:	1

Record an additional thread reaching the barrier.

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of GRF registers sent as the message payload. Value | Name | Description 1 | One **[Default]** | See MDP_Barrier Barrier Data Payload definition.
	24:20	**Response Length** Format: U5 Specifies the number of GRF registers expected as the message response payload. Value | Name | Description 0 | None **[Default]** | Barrier completion notification is signaled with ARF N0.0.
	19:16	**Reserved** Access: RO Format: MBZ
	15:12	**Reserved** Access: RO Format: MBZ
	11:3	**Reserved** Access: RO Format: MBZ
	2:0	**Barrier Subfunction** Default Value: 0x4 Format: OpCode

Bit Field Extract

bfe - Bit Field Extract

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

Component-wise extract a bit field from src2 using the bit field width from src0 and the bit field offset from src1. Store the extracted bit field value in the low bits of dst and sign extend (if D type) or zero extend (if UD type). The width and offset values are from the low five bits of src0 and src1 respectively, or src0 & 0x1f and src1 & 0x1f. If width is zero, the result is zero. If offset + width > 32 then the extracted bit field is bits offset to 31 of src2, extracting only 32 - offset bits, less than width as the bit field cannot extend past the MSB of the source value. Otherwise extract width bits extending from bit positions offset to offset + width - 1.

Format:

```
[(pred)] bfe (exec_size) dst src0 src1 src2
```

Restriction
No accumulator access, implicit or explicit.
All three-source instructions have certain restrictions, described in Instruction Formats.

Syntax

```
[(pred)] bfe (exec_size) reg reg reg reg
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         UD width = src0.chan[n][4:0];
         UD offset = src1.chan[n][4:0];
         if ( width == 0 ) {
             dst.chan[n] = 0x00000000;
         } else if ( (width + offset) < 32 ) {
             dst.chan[n] = src2.chan[n] « (32 - width - offset);
             if (src2 is signed) {
                 dst.chan[n] = dst.chan[n] » (32 - width); // pad sign bit of dst.chan
             } else {
                 dst.chan[n] = dst.chan[n] » (32 - width); // pad 0
             }
         } else {
             if ( src2 is signed ) {
                 dst.chan[n] = src2.chan[n] » offset; // pad sign bit
             } else {
                 dst.chan[n] = src2.chan[n] » offset; // pad 0
             }
         }
     }
 }
```

bfe - Bit Field Extract

Src Types	Dst Types
UD	UD
D	D

DWord	Bit	Description
0..3	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) AND ([Header][Opcode]!=madm) Format: **DirectOperand**
	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) AND ([Header][Opcode]==madm) Format: **MacroOperand**
	127:112	**Src2.ImmValue[15:0]** Exists If: ([Src2.IsImm]==true)
	113:112	**Src2.HorzStride** Exists If: ([Src2.IsImm]==false) Format: **HorzStride**
	111:98	**Src1.Operand** Exists If: ([Header][Opcode]!=madm) Format: **DirectOperand**
	111:98	**Src1.Operand** Exists If: ([Header][Opcode]==madm) Format: **MacroOperand**
	97:96	**Src1.HorzStride** Format: **HorzStride**
	95:92	**CondCtrl** Format: **FlagModifier**
	91	**Src1.VertStride[1]** Format: **TernaryVertStride[1:1]**
	90:88	**Src1.DataType** Format: **TernaryDataType**
	87:86	**Src1.Mod** Format: **SrcMod**
	85:84	**Src2.Mod** Format: **SrcMod**

<table>
<tr><th colspan="3">bfe - Bit Field Extract</th></tr>
<tr><td></td><td>83</td><td><b>Src1.VertStride[0]</b><table><tr><td>Format:</td><td>TernaryVertStride[0:0]</td></tr></table></td></tr>
<tr><td></td><td>82:80</td><td><b>Src2.DataType</b><table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false) AND ([Header][Opcode]!=madm)</td></tr><tr><td>Format:</td><td>DirectOperand</td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false) AND ([Header][Opcode]==madm)</td></tr><tr><td>Format:</td><td>MacroOperand</td></tr></table></td></tr>
<tr><td></td><td>79:64</td><td><b>Src0.ImmValue[15:0]</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>HorzStride</td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Header][Opcode]!=madm)</td></tr><tr><td>Format:</td><td>DirectOperand</td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Header][Opcode]==madm)</td></tr><tr><td>Format:</td><td>MacroOperand</td></tr></table></td></tr>
<tr><td></td><td>49</td><td><b>Reserved</b><table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>48</td><td><b>Dst.HorzStride</b>
This field provides the distance in unit of data elements between two adjacent data elements within a row (horizontal) in the register region for the operand.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>1 element</td></tr><tr><td>1</td><td>2 element</td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src2.IsImm</b>
This field indicate that Source 2 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
</table>

bfe - Bit Field Extract

<table>
<tr><td rowspan="12"></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td>SrcMod</td></tr></table></td></tr>
<tr><td>43</td><td><b>Src0.VertStride[1]</b><table><tr><td>Format:</td><td>TernaryVertStride[1:1]</td></tr></table></td></tr>
<tr><td>42:40</td><td><b>Src0.DataType</b><table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
<tr><td>39</td><td><b>ExecDataType</b>
This field indicate the datatype mode of ternary instruction. Integer or Float.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Integer</td></tr><tr><td>1</td><td>Float</td></tr></table></td></tr>
<tr><td>38:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
<tr><td>35</td><td><b>Src0.VertStride[0]</b><table><tr><td>Format:</td><td>TernaryVertStride[0:0]</td></tr></table></td></tr>
<tr><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td>30</td><td><b>Reserved</b></td></tr>
</table>

bfe - Bit Field Extract

29 **CmptCtrl**

Format:	MBZ

Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.

Value	Name	Description
0	NoCompaction **[Default]**	No compaction. 128-bit native instruction supporting all instruction options.
1	Compacted	Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.

28 **PredInv**

This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields

Value	Name	Description
0	Positive **[Default]**	Positive polarity of predication. Use the predication mask produced by PredCtrl.
1	Negative	Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.

27:24 **PredCtrl**

Format:	PredCtrl

This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.

23 **FlagRegNum[0]**

This field specifies bit[0] of the register number for a flag register operand.

22 **FlagSubRegNum**

This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.

21:19 **ChanOff**

Format:	ChanOff

This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.

bfe - Bit Field Extract

	18:16	**ExecSize** Format: ExecSize This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: Header

Bit Field Insert 1

bfi1 - Bit Field Insert 1

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The bfi1 instruction is the first instruction in a two-instruction macro for bfi (Bit Field Insert). The bfi1 instruction component-wise generates mask with control from src0 and src1 and stores the results in dst. The mask is used in the bfi2 instruction to generate the final result of bfi. Create a bit mask corresponding to the bit field width and offset in src0 and src1. Store the bit mask in dst. The mask has all bits in the bit field set to 1 and all other bits as 0. The width and offset values are from the low five bits of src0 and src1 respectively, or src0 & 0x1f and src1 & 0x1f. If width is zero, the result is zero. The bfi macro has four source operands: src0 - bit field width in low five bits, src1 - bit field offset/starting bit position in low five bits, src2 - bit field value to insert, using only the number of least significant bits given by width in src0, and src3 - overall value into which the bit field is inserted, providing all bits other than the inserted bits for the result value. bfi dst src0 src1 src2 src3 // Translates to these two instructions: bfi1 dst src0 src1 bfi2 dst dst src2 src3

Format:

```
        [(pred)] bfi1 (exec_size) dst src0 src1
```

Programming Notes

No accumulator access, implicit or explicit.

Syntax

```
[(pred)] bfi1 (exec_size) reg reg reg
 [(pred)] bfi1 (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
     for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         UD width = src0.chan[n][4:0];
         UD offset = src1.chan[n][4:0];
         dst = ((1 « width) - 1) « offset;
     }
 }
```

Src Types	Dst Types
UD	UD
D	D

DWord	Bit	Description

bfi1 - Bit Field Insert 1

0..3	127:126	**Reserved**	
		Exists If:	([Src1.IsImm]==false)
		Format:	MBZ
	127:96	**Src1.ImmValue[31:0]**	
		Exists If:	([Src1.IsImm]==true)
	125:122	**Reserved**	
		Exists If:	([Src1.IsImm]==false)
		Format:	MBZ
	121:120	**Src1.Mod**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**SrcMod**
	119:116	**Src1.VertStride**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**VertStride**
	115:113	**Src1.Width**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**Width**
	112	**Src1.AddrMode**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**AddrMode**
	111:98	**Src1.Operand**	
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
		Format:	**IndirectOperand**
	111:98	**Src1.Operand**	
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
		Format:	**DirectOperand**
	97:96	**Src1.HorzStride**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**HorzStride**
	95:92	**CondCtrl**	
		Format:	**FlagModifier**
	91:88	**Src1.DataType**	
		Exists If:	([Src1.IsImm]==true)
		Format:	**ImmDataType**

bfi1 - Bit Field Insert 1

<table>
<tr><td></td><td>91:88</td><td><b>Src1.DataType</b><table><tr><td>Exists If:</td><td>([Src1.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>87:84</td><td><b>Src0.VertStride</b><table><tr><td>Format:</td><td><b>VertStride</b></td></tr></table></td></tr>
<tr><td></td><td>83:81</td><td><b>Src0.Width</b><table><tr><td>Format:</td><td><b>Width</b></td></tr></table></td></tr>
<tr><td></td><td>80</td><td><b>Src0.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src1.IsImm</b>
This field indicate that Source 1 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
</table>

bfi1 - Bit Field Insert 1

<table>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td><b>Saturate</b></td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td><b>AccWrCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">bfi1 - Bit Field Insert 1</th></tr>
<tr><td></td><td></td><td><table><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td><b>Header</b></td></tr></table></td></tr>
</table>

Bit Field Insert 2

bfi2 - Bit Field Insert 2

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The bfi2 instruction is the second instruction in a two-instruction macro for bfi (Bit Field Insert). The bfi2 instruction component-wise performs the bitfield insert operation on src1 and src2 based on the mask in src0. Use the mask in src0 to take a bit field value from the low bits of src1 and combine it with the value from src2 (so src2 provides all bits other than those masked out and replaced by the bit field value). Store the result in dst. The bfi macro has four source operands: src0 - bit field width in low five bits, src1 - bit field offset/starting bit position in low five bits, src2 - bit field value to insert, using only the number of least significant bits given by width in src0, and src3 - overall value into which the bit field is inserted, providing all bits other than the inserted bits for the result value. bfi dst src0 src1 src2 src3 // Translates to these two instructions: bfi1 dst src0 src1 bfi2 dst dst src2 src3

Format:

```
      [(pred)] bfi2 (exec_size) dst src0 src1 src2
```

Restriction
No accumulator access, implicit or explicit.
All three-source instructions have certain restrictions, described in Instruction Formats.

Syntax

```
[(pred)] bfi2 (exec_size) reg reg reg reg
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         UD offset = LZD(reverse(src0.chan[n]))-1;
         // offset is the number of LSB zero bits below the bit mask which has all 1s.
         // width (implied by the logic) is the number of 1 bits in the mask value, which
should be all 1s.
         dst.chan[n] = ((src1.chan[n] « offset) & src0.chan[n]) | (src2.chan[n] & !
src0.chan[n]);
     }
 }
```

Src Types	Dst Types
UD	UD
D	D

DWord	Bit	Description

bfi2 - Bit Field Insert 2

0..3	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) AND ([Header][Opcode]!=madm) Format: **DirectOperand**
	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) AND ([Header][Opcode]==madm) Format: **MacroOperand**
	127:112	**Src2.ImmValue[15:0]** Exists If: ([Src2.IsImm]==true)
	113:112	**Src2.HorzStride** Exists If: ([Src2.IsImm]==false) Format: **HorzStride**
	111:98	**Src1.Operand** Exists If: ([Header][Opcode]!=madm) Format: **DirectOperand**
	111:98	**Src1.Operand** Exists If: ([Header][Opcode]==madm) Format: **MacroOperand**
	97:96	**Src1.HorzStride** Format: **HorzStride**
	95:92	**CondCtrl** Format: **FlagModifier**
	91	**Src1.VertStride[1]** Format: **TernaryVertStride[1:1]**
	90:88	**Src1.DataType** Format: **TernaryDataType**
	87:86	**Src1.Mod** Format: **SrcMod**
	85:84	**Src2.Mod** Format: **SrcMod**
	83	**Src1.VertStride[0]** Format: **TernaryVertStride[0:0]**
	82:80	**Src2.DataType** Format: **TernaryDataType**
	79:66	**Src0.Operand** Exists If: ([Src0.IsImm]==false) AND ([Header][Opcode]!=madm) Format: **DirectOperand**

bfi2 - Bit Field Insert 2

79:66 **Src0.Operand**

Exists If:	([Src0.IsImm]==false) AND ([Header][Opcode]==madm)
Format:	**MacroOperand**

79:64 **Src0.ImmValue[15:0]**

Exists If:	([Src0.IsImm]==true)

65:64 **Src0.HorzStride**

Exists If:	([Src0.IsImm]==false)
Format:	**HorzStride**

63:50 **Dst.Operand**

Exists If:	([Header][Opcode]!=madm)
Format:	**DirectOperand**

63:50 **Dst.Operand**

Exists If:	([Header][Opcode]==madm)
Format:	**MacroOperand**

49 **Reserved**

Format:	MBZ

48 **Dst.HorzStride**

This field provides the distance in unit of data elements between two adjacent data elements within a row (horizontal) in the register region for the operand.

Value	Name
0	1 element
1	2 element

47 **Src2.IsImm**

This field indicate that Source 2 operand is carrying an immediate value.

Value	Name
0	false
1	true

46 **Src0.IsImm**

This field indicate that Source 0 operand is carrying an immediate value.

Value	Name
0	false
1	true

45:44 **Src0.Mod**

Format:	**SrcMod**

43 **Src0.VertStride[1]**

Format:	**TernaryVertStride[1:1]**

<table>
<tr><th colspan="3">bfi2 - Bit Field Insert 2</th></tr>
<tr><td></td><td>42:40</td><td><b>Src0.DataType</b><table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
<tr><td></td><td>39</td><td><b>ExecDataType</b>
This field indicate the datatype mode of ternary instruction. Integer or Float.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Integer</td></tr><tr><td>1</td><td>Float</td></tr></table></td></tr>
<tr><td></td><td>38:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Src0.VertStride[0]</b><table><tr><td>Format:</td><td>TernaryVertStride[0:0]</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
</table>

bfi2 - Bit Field Insert 2

<table>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b>
<table><tr><td>Format:</td><td><b>Header</b></td></tr></table></td></tr>
</table>

Bit Field Reverse

bfrev - Bit Field Reverse

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The bfrev instruction component-wise reverses all the bits in src0 and stores the results in dst.

Format:

```
[(pred)] bfrev (exec_size) dst src0
```

Restriction

No accumulator access, implicit or explicit.

Syntax

```
[(pred)] bfrev (exec_size) reg reg
 [(pred)] bfrev (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         for ( idx = 0; idx < 32; idx++ ) {
             dst.chan[n][idx] = src0.chan[n][31-idx];
         }
     }
 }
```

Src Types	Dst Types
UD	UD

DWord	Bit	Description
0..3	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true)
	95:92	**CondCtrl** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **FlagModifier**
	95:64	**Src0.ImmValue[63:32]** Exists If: ([Src0.IsImm]==true) AND (([Src0.DataType]==:q) OR ([Src0.DataType]==:uq) OR ([Src0.DataType]==:df))

bfrev - Bit Field Reverse

Bits	Field		
87:84	**Src0.VertStride**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
		Format:	**VertStride**
83:81	**Src0.Width**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
		Format:	**Width**
80	**Src0.AddrMode**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
		Format:	**AddrMode**
79:66	**Src0.Operand**	Exists If:	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Direct)
		Format:	**DirectOperand**
79:66	**Src0.Operand**	Exists If:	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Indirect)
		Format:	**IndirectOperand**
65:64	**Src0.HorzStride**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
		Format:	**HorzStride**
63:50	**Dst.Operand**	Exists If:	([Dst.AddrMode]==Indirect)
		Format:	**IndirectOperand**
63:50	**Dst.Operand**	Exists If:	([Dst.AddrMode]==Direct)
		Format:	**DirectOperand**
49:48	**Dst.HorzStride**	Format:	**HorzStride**
47	**Reserved**	Access:	RO
		Format:	MBZ
46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value.		

<table>
<tr><th colspan="3">bfrev - Bit Field Reverse</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td>SrcMod</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td>ImmDataType</td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.</td></tr>
</table>

bfrev - Bit Field Reverse

Value	Name	Description
0	NoCompaction **[Default]**	No compaction. 128-bit native instruction supporting all instruction options.
1	Compacted	Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.

28 **PredInv**

This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields

Value	Name	Description
0	Positive **[Default]**	Positive polarity of predication. Use the predication mask produced by PredCtrl.
1	Negative	Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.

27:24 **PredCtrl**

Format:	**PredCtrl**

This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.

23 **FlagRegNum[0]**

This field specifies bit[0] of the register number for a flag register operand.

22 **FlagSubRegNum**

This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.

21:19 **ChanOff**

Format:	**ChanOff**

This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.

18:16 **ExecSize**

Format:	**ExecSize**

This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.

15:0 **Header**

Format:	**Header**

Boolean Function

bfn - Boolean Function

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	false
Source Modifier:	false
Syntax:	GROUP
Subfunctions:	BooleanFuncCtrl[95:92,87:84]

BFN performs an arbitrary boolean logical operation on three sources. Any of the three sources not involved in the boolean logical operation should have the null register supplied for that parameter.

Format:

```
[(pred)] bfn.<BooleanFuncCtrl> (exec_size) dst src0 src1 src2
```

<BooleanFuncCtrl> is a symbol defined in the BooleanFuncCtrl enumeration. This value indicates which of the 256 boolean functions to use and provides a full table of all the operations with some common equivalent boolean expressions that the function represents.

Determining the Function Index:
Given a desired boolean expression, one can derive the function index by combining special constants for each of the three sources in the expression with the four logical operations (NOT, AND, XOR, OR). Specifically let *s0* = 0xAA represent src0, *s1* = 0xCC represent src1, and *s2* = 0xF0 represent src2. Then combine these constants with any logical operations to get the function index for that logical combination.
For example, suppose we wanted s0|~s1&s2. This maps to (0xAA|~0xCC&0xF0) = **0xBA** and thus *bfn.0xBA*. A more complicated example might be *s0^~s1&~s2|s1&s2* (illustrating that sources may be used multiple times). This reduces to 0xAA^~0xCC&~0xF0|0xCC&0xF0 = **0xE9** and thus *bfn.0xE9*.

Programming Notes

EXAMPLES:

```
      bfn.0xEF (8) r10:ud  r0:ud  r1:ud   r2:ud    // computes r0|r1|~r2       bfn.0xFA
(8) r10:ud  r0:ud  null:ud r2:ud    // computes r0|r2
```

Restriction

All three-source instructions have certain restrictions, described in *Instruction Formats*

Source modifiers are illegal in this instruction, but unnecessary. If one desires the negation of an input, just select the corresponding function that enables that minor change. E.g. if one starts with wants a ternary OR (s0|s1|s2 or bfn.fFE), but desires to complement src2, then choose the function for (s0|s1|~s2).

Syntax

```
[(pred)] bfn.<BooleanFuncCtrl> (exec_size) reg reg reg reg
[(pred)] bfn.<BooleanFuncCtrl> (exec_size) reg reg reg imm16
[(pred)] bfn.<BooleanFuncCtrl> (exec_size) reg imm16 reg reg
```

bfn - Boolean Function

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         for ( idx = 0; idx < CHANNEL_SIZE_IN_BITS; idx++ ) { // foreach bit of dst, src0,
src1 and src2
             dst.chan[n][idx] = ( BooleanFuncCtrl » ((src0.chan[n][idx]) +
(src1.chan[n][idx] « 1) + (src2.chan[n][idx] « 2)) ) & 0x1;
         }
     }
 }
```

Src Types	Dst Types
UD, UW	UD, UW

DWord	Bit	Description
0..3	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) Format: **DirectOperand**
	127:112	**Src2.ImmValue[15:0]** Exists If: ([Src2.IsImm]==true)
	113:112	**Src2.HorzStride** Exists If: ([Src2.IsImm]==false) Format: **HorzStride**
	111:98	**Src1.Operand** Format: **DirectOperand**
	97:96	**Src1.HorzStride** Format: **HorzStride**
	95:92	**Lut8[7:4]** Format: **BooleanFuncCtrl[7:4]** These are bits[7:4] of lookup table lut8 of lop3 instruction.
	91	**Src1.VertStride[1]** Format: **TernaryVertStride[1:1]**
	90:88	**Src1.DataType** Format: **TernaryDataType**
	87:84	**Lut8[3:0]** Format: **BooleanFuncCtrl[3:0]** These are bits[3:0] of lookup table lut8 of lop3 instruction.
	83	**Src1.VertStride[0]** Format: **TernaryVertStride[0:0]**

bfn - Boolean Function

<table>
<tr><td rowspan="9"></td><td>82:80</td><td><b>Src2.DataType</b><table><tr><td>Format:</td><td><b>TernaryDataType</b></td></tr></table></td></tr>
<tr><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td>79:64</td><td><b>Src0.ImmValue[15:0]</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr></table></td></tr>
<tr><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>The Dst.Operand must be 64 bit aligned. i.e. Dst.Operand.SubRegNum[2:0] must be zero,</td></tr></table></td></tr>
<tr><td>49</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>48</td><td><b>Dst.HorzStride</b>
This field provides the distance in unit of data elements between two adjacent data elements within a row (horizontal) in the register region for the operand.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>1 element</td></tr><tr><td>1</td><td>2 element</td></tr></table></td></tr>
<tr><td>47</td><td><b>Src2.IsImm</b>
This field indicate that Source 2 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
</table>

bfn - Boolean Function

<table>
<tr><td></td><td>45:44</td><td><b>CondCtrl2</b>
A 2 bit compressed version of the FlagModifier.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>None <b>[Default]</b></td></tr>
<tr><td>01b</td><td>(ze)</td></tr>
<tr><td>10b</td><td>(gt)</td></tr>
<tr><td>11b</td><td>(lt)</td></tr>
</table></td></tr>
<tr><td></td><td>43</td><td><b>Src0.VertStride[1]</b>
<table><tr><td>Format:</td><td>TernaryVertStride[1:1]</td></tr></table></td></tr>
<tr><td></td><td>42:40</td><td><b>Src0.DataType</b>
<table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
<tr><td></td><td>39</td><td><b>ExecDataType</b>
This field indicate the datatype mode of ternary instruction. Integer or Float.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Integer</td></tr>
<tr><td>1</td><td>Float</td></tr>
</table></td></tr>
<tr><td></td><td>38:36</td><td><b>Dst.DataType</b>
<table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Src0.VertStride[0]</b>
<table><tr><td>Format:</td><td>TernaryVertStride[0:0]</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b>
<table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b>
<table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b>
<table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr>
<tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b>
<table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

bfn - Boolean Function

<table>
<tr><td></td><td></td><td>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td>PredCtrl</td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table>
<tr><td>Format:</td><td>ChanOff</td></tr>
</table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
</table>

bfn - Boolean Function

	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Branch Converging

brc - Branch Converging

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The brc instruction redirects the execution forward or backward to the instruction pointed by (current IP + offset). The jump will occur if all channels are branched away. UIP should reference the instruction where all channels are expected to come together. JIP should reference the end of the innermost conditional block.

In instruction binary, JIP and UIP use locations src1 and src0 respectively when immediate and location src0 when reg64, where reg64 is accessed as paired DWord (regioning being <2;2,1>). dst must be IP. When the offsets are immediate, src0 regfile must be immediate.

Format:

```
[(pred)] brc (exec_size) JIP UIP
```

Restriction
A brc instruction cannot use the Switch instruction option.

Syntax

```
[(pred)] brc (exec_size) imm32 imm32
 [(pred)] brc (exec_size) reg64
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < 32; n++ ) {
    if ( WrEn[n] ) {
       PcIP[n] = IP + UIP;
    } else {
       PcIP[n] = IP + 1;
    }
 }
 if ( all PcIP != IP + 1 ) {  // for all channels
    Jump(IP + JIP);
 }
```

DWord	Bit	Description	
0..3	127:96	**Reserved**	
		Exists If:	([Src0.IsImm]==false)
		Format:	MBZ

brc - Branch Converging

	127:96	**JIP** Exists If: ([Src0.IsImm]==true) Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:80	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	95:64	**Reserved** Exists If: ([Src0.IsImm]==true) AND ([Src1.IsImm]==false) Format: MBZ
	95:64	**UIP** Exists If: ([Src0.IsImm]==true) AND ([Src1.IsImm]==true) Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	79:66	**Src0.Operand** Exists If: ([Src0.IsImm]==false) Format: **DirectOperand**
	65:64	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	63:50	**Dst.Operand** Format: **DirectOperand**
	49:48	**Reserved** Access: RO Format: MBZ
	47	**Src1.IsImm** This field indicate that Source 1 operand is carrying an immediate value Value / Name 0: false 1: true
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value Value / Name 0: false 1: true

<table>
<tr><th colspan="3">brc - Branch Converging</th></tr>
<tr><td rowspan="8"></td><td>45:34</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>33</td><td><b>BranchCtrl</b>
This field is used by <i><b>goto</b></i>, <b>if</b>, and <i><b>else</b></i> instructions to control branching. See the <b>goto</b> instruction description for more information about BranchCtrl.</td></tr>
<tr><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td>30</td><td><b>Reserved</b></td></tr>
<tr><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
</table>

brc - Branch Converging

	27:24	**PredCtrl** Format: **PredCtrl** This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.
	23	**FlagRegNum[0]** This field specifies bit[0] of the register number for a flag register operand.
	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: **ChanOff** This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Branch Diverging

brd - Branch Diverging

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The brd instruction redirects the execution forward or backward to the instruction pointed by (current IP + offset). The jump will occur if any channels are branched away.

In instruction binary, JIP is at location src1 when immediate and at location src0 when reg32, where reg32 is accessed as a scalar DWord. The ip register must be used (for example, by the assembler) as dst.

Format:

```
         [(pred)] brd (exec_size) JIP
```

Restriction

A brd instruction cannot use the Switch instruction option.

Syntax

```
[(pred)] brd (exec_size) imm32
 [(pred)] brd (exec_size) reg32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < 32; n++ ) {
     if ( WrEn[n] ) {
         PcIP[n] = IP + JIP;
     } else {
         PcIP[n] = IP + 1;
     }
 }
 if ( any PcIP == ExIP + JIP ) {  // any channel
     Jump(ExIP + JIP);
 }
```

DWord	Bit	Description
0..3	127:96	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	127:96	**JIP** Exists If: ([Src0.IsImm]==true) Format: S31 The byte-aligned jump distance if a jump is taken for the channel

brd - Branch Diverging

95:80	**Reserved**			
		Exists If:	([Src0.IsImm]==false)	
		Format:	MBZ	
95:64	**Reserved**			
		Exists If:	([Src0.IsImm]==true)	
		Format:	MBZ	
79:66	**Src0.Operand**			
		Exists If:	([Src0.IsImm]==false)	
		Format:	**DirectOperand**	
65:64	**Reserved**			
		Exists If:	([Src0.IsImm]==false)	
		Format:	MBZ	
63:50	**Dst.Operand**			
		Format:	**DirectOperand**	
49:47	**Reserved**			
		Access:	RO	
		Format:	MBZ	
46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value.			
		Value	**Name**	
		0	false	
		1	true	
45:34	**Reserved**			
		Access:	RO	
		Format:	MBZ	
33	**BranchCtrl** This field is used by ***goto*** ,**if**, and ***else*** instructions to control branching. See the **goto** instruction description for more information about BranchCtrl.			
32	**AtomicCtrl**			
		Format:	**AtomicCtrl**	
31	**MaskCtrl** Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".			
		Value	**Name**	**Description**
		0	Normal **[Default]**	Normal. Per channel write enable used for final write enable generation.
		1	NoMask	NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.

<table>
<tr><th colspan="3">brd - Branch Diverging</th></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b>
<table><tr><td>Format:</td><td>MBZ</td></tr></table>
Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
</table>

brd - Branch Diverging

	21:19	**ChanOff** Format: **ChanOff** This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Break

break - Break

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The break instruction is used to early-out from the inner most loop, or early out from the inner most switch block. When used in a loop, upon execution, the break instruction terminates the loop for all execution channels enabled. If all the enabled channels hit the break instruction, jump to the instruction referenced by JIP. JIP should be the offset to the end of the inner most conditional or loop block, UIP should be the offset to the while instruction of the loop block. If SPF is ON, the UIP must be used to update IP; JIP is not used in this case

The following table describes the two 32-bit instruction pointer offsets. Both the JIP and UIP are signed 32-bit numbers, added to IP pre-increment. In instruction binary, JIP and UIP are at locations src0 and src1 and must be of type DW (signed DWord integer). When the offsets are immediate, src0 regfile must be immediate.

Format:

```
[(pred)] break (exec_size) JIP UIP
```

Syntax

```
[(pred)] break (exec_size) imm32 imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.channel[n] ) {
         PcIP[n] = IP + UIP;
     } else {
         PcIP[n] = IP + 1;
     }
 }
 if ( PcIP != (IP + 1) ) { // all channels
     Jump(IP + JIP);
 }
```

DWord	Bit	Description
0..3	127:96	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	127:96	**JIP** Exists If: ([Src0.IsImm]==true) Format: S31 The byte-aligned jump distance if a jump is taken for the channel.

<table>
<tr><th colspan="3">break - Break</th></tr>
<tr><td></td><td>95:80</td><td><b>Reserved</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>95:64</td><td><b>Reserved</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true) AND ([Src1.IsImm]==false)</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>95:64</td><td><b>UIP</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true) AND ([Src1.IsImm]==true)</td></tr><tr><td>Format:</td><td>S31</td></tr></table>The byte aligned jump distance if a jump is taken for the instruction.</td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Reserved</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src1.IsImm</b>
This field indicate that Source 1 operand is carrying an immediate value<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:34</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>BranchCtrl</b>
This field is used by <b><i>goto</i></b>, <b>if</b>, and <b><i>else</i></b> instructions to control branching. See the <b>goto</b> instruction description for more information about BranchCtrl.</td></tr>
</table>

<table>
<tr><th colspan="3">break - Break</th></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr>
<tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td>PredCtrl</td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
</table>

break - Break

	23	**FlagRegNum[0]** This field specifies bit[0] of the register number for a flag register operand.
	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: **ChanOff** This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

BTD Spawn Message MSD

BTD_SPAWN_MSD - BTD Spawn Message MSD

Source:	SFID_BTD
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15
	24:20	**Reserved** Access: RO Format: MBZ
	19	**Header Present** Format: Enable **Programming Notes** Must be programmed to 0
	18	**Reserved** Access: RO Format: MBZ
	17:14	**Message Type** Default Value: 01h Format: Opcode Bindless Thread Dispatch (BTD) Spawn Message
	13:9	**Reserved** Access: RO Format: MBZ
	8	**SIMD mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Reserved** Access: RO Format: MBZ

Byte Scattered Read MSD

MSD0R_BS - Byte Scattered Read MSD

Source:	EuSubFunctionDataPort0
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message

MSD0R_BS - Byte Scattered Read MSD

	17:14	**Message Type** Default Value: 04h Format: Opcode Byte Scattered Read message
	13	**Reserved** Access: RO Format: MBZ
	12	**Reserved** Access: RO Format: MBZ
	11:10	**Data Elements** Format: MDC_DS Specifies the number of Bytes to be read or written per Dword
	9	**Reserved** Access: RO Format: MBZ
	8	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Byte Scattered Write MSD

MSD0W_BS - Byte Scattered Write MSD

Source:	EuSubFunctionDataPort0
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message

MSD0W_BS - Byte Scattered Write MSD

	17:14	**Message Type** Default Value: 0Ch Format: Opcode Byte Scattered Write message
	13:12	**Reserved** Access: RO Format: MBZ
	11:10	**Data Elements** Format: **MDC_DS** Specifies the number of Bytes to be read or written per Dword
	9	**Reserved** Access: RO Format: MBZ
	8	**SIMD Mode** Format: **MDC_SM2** Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Binding Table Index** Format: **MDC_BTS_SLM_A32** Specifies the Binding Table Index for the message

Call

call - Call

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The call instruction jumps to a subroutine. It can be predicated or non-predicated. If non-predicated, all enabled channels jump to the subroutine. If predicated, only the channels enabled by PMask jump to the subroutine; the rest of the channels move to the next instruction after the call instruction. If none of the channels jump into the subroutine, the call instruction is treated as a nop. In case of a jump, the call instruction stores the return IP onto the first DWord of the destination register and stores the CallMask in the second DWord of the destination register. When SPF is on, the predication control must be scalar.

The following section describes JIP, the jump offset.
JIP can be an immediate or register value. When a jump occurs, this value is added to IP pre-increment. In instruction binary, JIP is at location src1 and src0 must be null. The GRF register must be put (for example, by the assembler) at dst location.
Format: [(pred)] call (exec_size) dst JIP

Format:

```
[(pred)] call (exec_size) dst JIP
```

Restriction
The call instruction must have DWord source and destination type, and the destination must be QWord aligned.
A call instruction must target an instruction with Switch set; in addition, the instruction following the call must also have Switch set.
When EU Fusion is enabled JIP of both EU's must be same.

Syntax

```
[(pred)] call (exec_size) reg imm32 [(pred)] call (exec_size) reg reg32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if (WrEn.chan[n] ) {
         PcIP[n] = IP + JIP;
         CallMask[n] = 1;
     } else {
         PcIP[n] = IP + 1;
         CallMask[n] = 0;
     }
 }
 if ( PcIP[n] != (IP + 1) ) {  // any channel jumped
```

call - Call

```
     dst.chan[0] = IP + 1;
     dst.chan[1] = CallMask;
     Jump(IP + JIP);
}
```

DWord	Bit	Description
0..3	127:96	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	127:96	**JIP** Exists If: ([Src0.IsImm]==true) Format: S31 The byte-aligned jump distance if a jump is taken for the channel
	95:80	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	95:64	**Reserved** Exists If: ([Src0.IsImm]==true) Format: MBZ
	79:66	**Src0.Operand** Exists If: ([Src0.IsImm]==false) Format: DirectOperand
	65:64	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	63:50	**Dst.Operand** Format: DirectOperand
	49:47	**Reserved** Access: RO Format: MBZ
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value. Value / Name: 0 – false 1 – true
	45:34	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">call - Call</th></tr>
<tr><td></td><td>33</td><td><b>BranchCtrl</b>
This field is used by <b><i>goto</i></b>, <b>if</b>, and <b><i>else</i></b> instructions to control branching. See the <b>goto</b> instruction description for more information about BranchCtrl.</td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b>
<table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr>
<tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b>
<table><tr><td>Format:</td><td>MBZ</td></tr></table>
Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
</table>

call - Call		
	27:24	**PredCtrl** Format: **PredCtrl** This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.
	23	**FlagRegNum[0]** This field specifies bit[0] of the register number for a flag register operand.
	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: **ChanOff** This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Call Absolute

calla - Call Absolute

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The calla instruction jumps to a subroutine. It can be predicated or non-predicated. If non-predicated, all enabled channels jump to the subroutine. If predicated, only the channels enabled by PMask jump to the subroutine; the rest of the channels move to the next instruction after the calla instruction. If none of the channels jump into the subroutine, the calla instruction is treated as a nop. In case of a jump, the call instruction stores the return IP onto the first DWord of the destination register and stores the CallMask in the second DWord of the destination register. If SPF is ON, none of the PcIP are updated. When SPF is on, the predication control must be scalar. The difference between calla and call is that calla uses JIP as the IP value rather than adding it to the IP value.

Format:

```
[(pred)] calla (exec_size) dst JIP
```

Restriction
The calla instruction must have DWord source and destination type, and the destination must be QWord-aligned.
When EU Fusion is enabled JIP of both EU's must be same.

Syntax
`[(pred)] calla (exec_size) reg imm32`
`[(pred)] calla (exec_size) reg reg32`

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.channel[n] ) {
         PcIP[n] = JIP;
         CallMask[n] = 1;
     } else {
         PcIP[n] = IP + 1;
         CallMask[n] = 0;
     }
 }
 if ( PcIP[n] != (IP + 1) ) {  // any channel jumped
     dst.chan[0] = IP + 1;
     dst.chan[1] = CallMask;
     Jump(JIP);
 }
```

DWord	Bit	Description

calla - Call Absolute

0..3	127:96	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	127:96	**JIP** Exists If: ([Src0.IsImm]==true) Format: S31 The byte-aligned jump distance if a jump is taken for the channel
	95:80	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	95:64	**Reserved** Exists If: ([Src0.IsImm]==true) Format: MBZ
	79:66	**Src0.Operand** Exists If: ([Src0.IsImm]==false) Format: **DirectOperand**
	65:64	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	63:50	**Dst.Operand** Format: **DirectOperand**
	49:47	**Reserved** Access: RO Format: MBZ
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value. **Value** | **Name** 0 | false 1 | true
	45:34	**Reserved** Access: RO Format: MBZ
	33	**BranchCtrl** This field is used by ***goto***, **if**, and ***else*** instructions to control branching. See the **goto** instruction description for more information about BranchCtrl.
	32	**AtomicCtrl** Format: **AtomicCtrl**

calla - Call Absolute

<table>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr>
<tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b>
<table>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td>PredCtrl</td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
</table>

calla - Call Absolute

	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: **ChanOff** This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

CCS Page Fast Clear

DP_CCS_PAGE_CLEAR - CCS Page Fast Clear

Source:	SFID_1, SFID_F
Length Bias:	1

Updates the compression metadata to clear the 64KB page containing the specified address.

Programming Notes
The src0 address payload specifies the page address.
The src1 data payload is null.

Restriction
Setting a page to "clear" state is not allowed if the surface is accessed as Untyped compressed buffer.

Syntax

```
[(pred)] CCS_PAGE_CLEAR.sfid (exec_mask) <addr_type[+offset]>src0_reg:addr_size
src1_nullreg
```

Pseudocode

```
CCS_PAGE_UPDATE((Base+offset)+(src0.addr)) = CLEAR;
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Must be FLAT.
	28:25	**Src0 Length** Format: DP_ONE_ADDR_REG Specifies the size of the address payload, in registers. Address payload format is A64_PAYLOAD_SIMT1.
	24:20	**Dest Length** Format: U5 Specifies the size of destination data register payload.

Value	Name	Description
0		No data returned in registers.

DP_CCS_PAGE_CLEAR - CCS Page Fast Clear

	19:17	**Fast Clear** Default Value: 0 Format: Opcode
	16:9	**Reserved** Access: RO Format: MBZ
	8:7	**Address Size** Format: DP_ADDR_SIZE Specifies the bit size of the address payload item. **Restriction** Must be A64.
	6	**Reserved** Access: RO Format: MBZ
	5:0	**CCS Update** Default Value: 29 Format: Opcode

CCS Page Fast Uncompress

DP_CCS_PAGE_UNCOMPRESS - CCS Page Fast Uncompress

Source:	SFID_1, SFID_F
Length Bias:	1

Updates the compression metadata to uncompress the 64KB page with the specified address.

Programming Notes
The src0 address payload specifies the page address.
The src1 data payload is null.

Syntax

```
[(pred)] CCS_PAGE_CLEAR.sfid (exec_mask) <addr_type[+offset]>src0_reg:addr_size
src1_nullreg
```

Pseudocode

```
CCS_PAGE_UPDATE((Base+offset)+(src0.addr)) = UNCOMPRESS;
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Must be FLAT.
	28:25	**Src0 Length** Format: DP_ONE_ADDR_REG Specifies the size of the address payload, in registers. Address payload format is A64_PAYLOAD_SIMT1.
	24:20	**Dest Length** Format: U5 Specifies the size of destination data register payload. Value: 0 — Name: — Description: No data returned in registers.
	19:17	**Fast Uncompress** Default Value: 2 Format: Opcode

DP_CCS_PAGE_UNCOMPRESS - CCS Page Fast Uncompress

	16:9	**Reserved** Access: RO Format: MBZ
	8:7	**Address Size** Format: **DP_ADDR_SIZE** Specifies the bit size of the address payload item. **Restriction** Must be A64.
	6	**Reserved** Access: RO Format: MBZ
	5:0	**CCS Update** Default Value: 29 Format: Opcode

CCS Sector Slow Clear

DP_CCS_SEC_CLEAR - CCS Sector Slow Clear

Source:	SFID_D
Length Bias:	1

For each enabled SIMT lane, the L3 sector compression metadata is set to cleared.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload is null.

Restriction
Setting a sector to "clear" state is not allowed if the surface is accessed as Untyped compressed buffer.

Syntax

```
[(pred)] CCS_SEC_CLEAR.sfid (exec_mask) <addr_type[+offset]>src0_reg:addr_size
src1_nullreg:data_size[.vect_size]
```

Pseudocode

```
dsize = Typed ? SURFACE_STATE.element_size : data_size; for (n = 0; n < 32; n++)
{ if (Msg.ChEn[n]) { for (m = v = 0; v < 4; v++) { if (cmask[v]) {
CCS_SECTOR_CLEAR((Base+offset)+(src0.addr_size[n])).dsize); m++; } } } }
```

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>30:29</td><td><b>Address Type</b><table><tr><td>Format:</td><td>DP_ADDR_SURFACE_TYPE</td></tr></table>Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc.<table><tr><th>Restriction</th></tr><tr><td>This message is not allowed on SCRATCH surfaces.</td></tr></table></td></tr>
<tr><td></td><td>28:25</td><td><b>Src0 Length</b><table><tr><td>Format:</td><td>DP_ADDR_REG_SIZE</td></tr></table>Specifies the size of the address payload, in registers.</td></tr>
<tr><td></td><td>24:20</td><td><b>Dest Length</b><table><tr><td>Format:</td><td>U5</td></tr></table>Specifies the size of destination data register payload.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td></td><td>No data returned in registers.</td></tr></table></td></tr>
</table>

DP_CCS_SEC_CLEAR - CCS Sector Slow Clear

	19:17	**Slow Clear** Default Value: 1 Format: Opcode
	16	**Reserved** Access: RO Format: MBZ
	15:12	**Component Mask** Format: **DP_CMASK** Specifies the component mask of each data payload item.
	11:9	**Reserved** Access: RO Format: MBZ
	8:7	**Address Size** Format: **DP_ADDR_SIZE** Specifies the bit size of each address payload item.
	6	**Reserved** Access: RO Format: MBZ
	5:0	**CCS Update** Default Value: 29 Format: Opcode

CCS Sector Slow Uncompress

DP_CCS_SEC_UNCOMPRESS - CCS Sector Slow Uncompress

Source:	SFID_D
Length Bias:	1

For each enabled SIMT lane, the L3 sector compression metadata is set to uncompressed.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload is null.

Syntax

```
[(pred)] CCS_SEC_UNCOMPRESS.sfid (exec_mask) <addr_type[+offset]>src0_reg:addr_size
src1_nullreg:data_size[.vect_size]
```

Pseudocode

```
dsize = Typed ? SURFACE_STATE.element_size : data_size;
for (n = 0; n < 32; n++) {
    if (Msg.ChEn[n]) {
        for (m = v = 0; v < 4; v++) {
            if (cmask[v]) {
                CCS_SECTOR_UNCOMPRESS((Base+offset)+(src0.addr_size[n])).dsize);
                m++;
            }
        }
    }
}
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** This message is not allowed on SCRATCH surfaces.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.
	24:20	**Dest Length** Format: U5 Specifies the size of destination data register payload. Value | Name | Description 0 | | No data returned in registers.
	19:17	**Slow Uncompress** Default Value: 3 Format: Opcode

DP_CCS_SEC_UNCOMPRESS - CCS Sector Slow Uncompress

	16	**Reserved** Access: RO Format: MBZ
	15:12	**Component Mask** Format: **DP_CMASK** Specifies the component mask of each data payload item.
	11:9	**Reserved** Access: RO Format: MBZ
	8:7	**Address Size** Format: **DP_ADDR_SIZE** Specifies the bit size of each address payload item.
	6	**Reserved** Access: RO Format: MBZ
	5:0	**CCS Update** Default Value: 29 Format: Opcode

CFE_STATE

CFE_STATE - CFE_STATE

Source:	RenderCS, ComputeCS
Length Bias:	2

Set the compute pipeline state.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Compute Format: OpCode
	26:24	**Compute Command Opcode** Default Value: 2h New CFE Command Format: OpCode
	23:18	**CFE SubOpcode** Default Value: 0h CFE_STATE Format: OpCode
	17:16	**CFE SubOpcode Variant** Default Value: 0 Standard Format: U2
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Format: =n Value | Name | Description 04h | DWORD_COUNT_n **[Default]** | Excludes DWord (0,1)
1..2	63:32	**Reserved** Access: RO Format: MBZ
	31:10	**Scratch Space Buffer** Format: SurfaceStateOffset[27:6] Specifies the surface state index to the Scratch Buffer for use by the kernel. This surface state index is relative to the **Surface State Base Address**.

<table>
<tr><th colspan="3">CFE_STATE - CFE_STATE</th></tr>
<tr><td></td><td></td><td><table><tr><th>Programming Notes</th></tr><tr><td>The driver must allocate the Scratch Buffer surface to ensure that all threads have their own per-thread scratch space.
(Pitch = Per Thread Scratch Space, Number of entries in the buffer = Maximum Number Of Threads.)</td></tr></table></td></tr>
<tr><td></td><td>9:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>3</td><td>31:16</td><td><b>Maximum Number of Threads</b><table><tr><td>Format:</td><td>U16</td></tr></table><table><tr><th>Description</th></tr><tr><td>Range: [1, 2^16-1], representing [1, 2^16-1] threads.
Normally set to the maximum number of threads: (# EUs) * (# threads/EU). See <i>Graphics Processing Engine</i> for listing of #EUs and #threads in each device.
See Programming Restrictions here for additional limitations.</td></tr><tr><td>Restriction :
The smallest number of maximum threads supported is 64.
The largest number may exceed the number of threads on the GPU, but must be <= (#Slices)*1024. (Range of FFTID per slice is 10 bits.)</td></tr></table></td></tr>
<tr><td></td><td>15:14</td><td><b>Over Dispatch Control</b><table><tr><td>Format:</td><td>U2</td></tr></table>Enables the amount of GPGPU thread over dispatch.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>None</td><td>0% overdispatch</td></tr><tr><td>1</td><td>Low</td><td>25% overdispatch</td></tr><tr><td>2</td><td>Normal <b>[Default]</b></td><td>50% overdispatch</td></tr><tr><td>3</td><td>High</td><td>75% overdispatch</td></tr></table></td></tr>
<tr><td></td><td>13</td><td><b>Single Slice Dispatch CCS Mode</b><table><tr><td>Format:</td><td>Enable</td></tr></table>Specifies whether thread dispatches from this CCS context are limited to the single physical C-slice that is used when all CCS are active. Only active when RCU Load Balance Slice CCS Mode is set.
When this mode set, any other C-slice that has been assigned to this CCS context by load balancing is not used by this CCS context thread dispatcher. When clear, all C-slices assigned to the CCS context by load-balancing are used by this CCS context thread dispatcher.
Ignored by RCS context thread dispatcher.</td></tr>
</table>

CFE_STATE - CFE_STATE

Value	Name
0	Disabled **[Default]**
1	Enabled

Programming Notes
When multiple CCS contexts are being used in load balance mode, and the submitted work runs most efficiently within a single C-slice, then this Single Slice Dispatch mode enables other CCS submissions to start faster.

12 **Reserved**

Format:	MBZ

11 **Reserved**

Format:	MBZ

11 **Compute Overdispatch Disable**

Format:	Disable

When this bit is set, the thread dispatch logic will disable over dispatching of threads to the DSS.

Value	Name
0	Enabled **[Default]**
1	Disabled

Programming Notes
SW must only set this bit for long running kernels, else there is a risk of performance degradation such as reduced IPC.

10 **Reserved**

10 **Reserved**

Format:	MBZ

9 **Reserved**

Format:	MBZ

9 **Local ID Tile Y optimization enable**

Format:	Enable

If this bit is set, the Tile Y Local ID optimization is enabled.

Value	Name
0b	Disabled **[Default]**
1b	Enabled

8:7 **Reserved**

Access:	RO
Format:	MBZ

CFE_STATE - CFE_STATE

<table>
<tr><td></td><td>6</td><td><b>Fused EU Dispatch</b>
<table>
<tr><td>Default Value:</td><td>0h Fused EU Mode</td></tr>
<tr><td>Format:</td><td>Disable</td></tr>
</table>
This field determine if threads will be dispatched in sets to fused EUs if set or if they will be dispatched individually. Depending on the project, the set size can be 2 or 4. If dispatched in sets the fused threads will all be part of the same thread group for GPGPU threads or will be part of the same iteration of the inner local loop if media threads.</td></tr>
<tr><td></td><td>5:3</td><td><b>Number of Walkers</b>
<table>
<tr><td>Format:</td><td>U3-1</td></tr>
</table>
Number of walkers (minus one) simultaneously supported by the COMPUTE WALKER command.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Programming Notes</th></tr>
<tr><td>[0,1]</td><td></td><td>One or two active walkers per context.</td><td>Value is ignored. Walkers are always processed in order.</td></tr>
</table></td></tr>
<tr><td></td><td>2:0</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>4</td><td>31:0</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>5</td><td>31:11</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>10:1</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>0</td><td><b>Reserved</b></td></tr>
</table>

Compare

cmp - Compare

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	false
Source Modifier:	true

The cmp instruction performs component-wise comparison of src0 and src1 and stores the results in the selected flag register and in dst. It takes component-wise subtraction of src0 and src1, evaluating the conditional code (excluding NS signal) based on the conditional modifier, and storing the conditional bits in bit-packed form in the destination flag register and all bits of dst channels. If the dst is not null, for the enabled channels, then all bits of the destination channel will contain the flag value for the channel. When the instruction operates on packed word format, one general register may store up to 16 such comparison results. In DWord format, one general register may store up to 8 results. A conditional modifier must be specified; the conditional modifier field cannot be 0000b. The comparison does not use the NS (NaN source) signals, as described in the Creating Conditional Flags section. Accordingly the conditional modifier should not be .u (unordered). For each enabled channel 0b or 1b is assigned to the appropriate flag bit and 0/all zeros or all ones (e.g, byte 0xFF, word 0xFFFF, DWord 0xFFFFFFFF) is assigned to dst. When any source type is floating-point, the cmp instruction obeys the rules described in the tables in the Floating Point Modes section of the Data Types chapter.
Refer to **Floating-Point Compare Operations** and **Assigning Conditional Flags** for details.

Format:

```
[(pred)] cmp[.cmod] (exec_size) dst src0 src1
```

Restriction

Pure bfloat operation is not supported.

Syntax

```
[(pred)] cmp[.cmod] (exec_size) reg reg reg
 [(pred)] cmp[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         results[n] = src0.chan[n] - src1.chan[n];
         bitMask[n] = Condition(results[n]); // details in Assigning Conditional Flags
dst.chan[n] = bitMask[n]; // All bits for dst channel
         flag#.bit[n]= bitMask[n];
     }
 }
```

Src Types	**Dst Types**

cmp - Compare

*B,*W,*D	*B,*W,*D
F	F
HF	HF
BF, F	BF, F

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)
	125:122	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	121:120	**Src1.Mod** Exists If: ([Src1.IsImm]==false) Format: **SrcMod**
	119:116	**Src1.VertStride** Exists If: ([Src1.IsImm]==false) Format: **VertStride**
	115:113	**Src1.Width** Exists If: ([Src1.IsImm]==false) Format: **Width**
	112	**Src1.AddrMode** Exists If: ([Src1.IsImm]==false) Format: **AddrMode**
	111:98	**Src1.Operand** Exists If: ([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect) Format: **IndirectOperand**
	111:98	**Src1.Operand** Exists If: ([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct) Format: **DirectOperand**
	97:96	**Src1.HorzStride** Exists If: ([Src1.IsImm]==false) Format: **HorzStride**
	95:92	**CondCtrl** Format: **FlagModifier**

cmp - Compare

	91:88	**Src1.DataType** Exists If: ([Src1.IsImm]==true) Format: **ImmDataType**
	91:88	**Src1.DataType** Exists If: ([Src1.IsImm]==false) Format: **RegDataType**
	87:84	**Src0.VertStride** Format: **VertStride**
	83:81	**Src0.Width** Format: **Width**
	80	**Src0.AddrMode** Format: **AddrMode**
	79:66	**Src0.Operand** Exists If: ([Src0.AddrMode]==Direct) Format: **DirectOperand**
	79:66	**Src0.Operand** Exists If: ([Src0.AddrMode]==Indirect) Format: **IndirectOperand**
	65:64	**Src0.HorzStride** Format: **HorzStride**
	63:50	**Dst.Operand** Exists If: ([Dst.AddrMode]==Direct) Format: **DirectOperand**
	63:50	**Dst.Operand** Exists If: ([Dst.AddrMode]==Indirect) Format: **IndirectOperand**
	49:48	**Dst.HorzStride** Format: **HorzStride**
	47	**Src1.IsImm** This field indicate that Source 1 operand is carrying an immediate value. Value | Name 0 | false **[Default]** 1 | true
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value.

<table>
<tr><th colspan="3">cmp - Compare</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td>SrcMod</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td>ImmDataType</td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.</td></tr>
</table>

cmp - Compare

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table>
</td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table>
</td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td><b>PredCtrl</b></td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table>
<tr><td>Format:</td><td><b>ChanOff</b></td></tr>
</table>
This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table>
<tr><td>Format:</td><td><b>ExecSize</b></td></tr>
</table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b>
<table>
<tr><td>Format:</td><td><b>Header</b></td></tr>
</table>
</td></tr>
</table>

Compare NaN

cmpn - Compare NaN

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	false
Source Modifier:	true

The cmpn instruction performs component-wise special-NaN comparison of src0 and src1 and stores the results in the selected flag register and in dst. It takes component-wise subtraction of src0 and src1, evaluating the conditional signals including NS based on the conditional modifier, and storing the conditional flag bits in bit-packed form in the destination flag register and all bits of dst channels. If the dst is not null, for the enabled channels, then all bits of the destination channel will contain the flag value for the channel. When the instruction operates on packed word format, one general register may store up to 16 such comparison results. In DWord format, one general register may store up to 8 results. A conditional modifier must be specified; the conditional modifier field cannot be 0000b. More information about the conditional signals used is in the Creating Conditional Flags section. For each enabled channel 0b or 1b is assigned to the appropriate flag bit and 0/all zeros or all ones (e.g, byte 0xFF, word 0xFFFF, DWord 0xFFFFFFFF) is assigned to dst. Min/Max instructions use cmpn to select the destination from the input sources (see the Min Max of Floating Point Numbers section for details).
Refer to **Floating-Point Compare Operations** and **Assigning Conditional Flags** for details.

Format:

```
[(pred)] cmpn[.cmod] (exec_size) dst src0 src1
```

Restriction

.l and .ge are the only two conditional modifiers are supported for this instruction.

Syntax

```
[(pred)] cmpn[.cmod] (exec_size) reg reg reg
 [(pred)] cmpn[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         results[n] = src0.chan[n] - src1.chan[n];
         bitMask[n] = ConditionNaN(results[n]); // details in Assigning Conditional Flags
dst.chan[n][0] = bitMask[n]; // All bits for dst channel
         flag#.bit[n] = bitMask[n];
     }
 }
```

Src Types	Dst Types
*B,*W,*D	*B,*W,*D

cmpn - Compare NaN

F	F
HF	HF

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)
	125:122	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	121:120	**Src1.Mod** Exists If: ([Src1.IsImm]==false) Format: **SrcMod**
	119:116	**Src1.VertStride** Exists If: ([Src1.IsImm]==false) Format: **VertStride**
	115:113	**Src1.Width** Exists If: ([Src1.IsImm]==false) Format: **Width**
	112	**Src1.AddrMode** Exists If: ([Src1.IsImm]==false) Format: **AddrMode**
	111:98	**Src1.Operand** Exists If: ([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect) Format: **IndirectOperand**
	111:98	**Src1.Operand** Exists If: ([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct) Format: **DirectOperand**
	97:96	**Src1.HorzStride** Exists If: ([Src1.IsImm]==false) Format: **HorzStride**
	95:92	**CondCtrl** Format: **FlagModifier**

cmpn - Compare NaN

	91:88	**Src1.DataType** Exists If: | ([Src1.IsImm]==true) Format: | **ImmDataType**
	91:88	**Src1.DataType** Exists If: | ([Src1.IsImm]==false) Format: | **RegDataType**
	87:84	**Src0.VertStride** Format: | **VertStride**
	83:81	**Src0.Width** Format: | **Width**
	80	**Src0.AddrMode** Format: | **AddrMode**
	79:66	**Src0.Operand** Exists If: | ([Src0.AddrMode]==Direct) Format: | **DirectOperand**
	79:66	**Src0.Operand** Exists If: | ([Src0.AddrMode]==Indirect) Format: | **IndirectOperand**
	65:64	**Src0.HorzStride** Format: | **HorzStride**
	63:50	**Dst.Operand** Exists If: | ([Dst.AddrMode]==Direct) Format: | **DirectOperand**
	63:50	**Dst.Operand** Exists If: | ([Dst.AddrMode]==Indirect) Format: | **IndirectOperand**
	49:48	**Dst.HorzStride** Format: | **HorzStride**
	47	**Src1.IsImm** This field indicate that Source 1 operand is carrying an immediate value. **Value** | **Name** 0 | false **[Default]** 1 | true
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value.

<table>
<tr><th colspan="3">cmpn - Compare NaN</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td>SrcMod</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td>ImmDataType</td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.</td></tr>
</table>

cmpn - Compare NaN

<table>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td>PredCtrl</td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td>ChanOff</td></tr></table>This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td>ExecSize</td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td>Header</td></tr></table></td></tr>
</table>

COMPUTE_WALKER

COMPUTE_WALKER - COMPUTE_WALKER

Source:	RenderCS, ComputeCS
Length Bias:	2

COMPUTE_WALKER spawns threadgroups in 1, 2, or 3 dimensions (X, Y, Z). Each threadgroup is described by Interface Descriptor in this command.
Each dispatched thread has a standard payload delivered in R0, including the Indirect Address to fetch the thread's parameters.
After the Walker completes dispatching its threads and those threads have completed running, a PostSync operation can write a completion code or a timestamp.

Programming Notes
If the threads spawned by this command are required to observe memory writes performed by threads spawned from a previous command, and if those threads did not perform a Memory Fence before they exited, then software must precede this command with a PIPE_CONTROL with "HDC Pipeline Flush control" plus "Untyped L1 cache flush" bits set.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Compute Format: OpCode
	26:24	**Compute Command Opcode** Default Value: 2h New CFE Command Format: OpCode
	23:18	**CFE SubOpcode** Default Value: 2 COMPUTE_WALKER Format: Opcode
	17:16	**CFE SubOpcode Variant** Default Value: 0 Standard Format: U2
	15	**Reserved** Access: RO Format: MBZ
	14	**Systolic Mode Enable** Format: Enable This bit specifies whether systolic mode is enabled or not. This field is overwritten by the hardware based on the pipeline select systolic mode. This is

COMPUTE_WALKER - COMPUTE_WALKER

		required as part of the thread dispatch to ensure systolic array operations are only executed when systolic mode is enabled.
	13:11	**Reserved** Access: RO Format: MBZ
	10	**Indirect Parameter Enable** Format: Enable If set, the Dimension X/Y/Z values in DW 7/8/9 are ignored and replaced by the current values of the corresponding GPGPU_xxx MMIO registers: • GPGPU_DISPATCHDIMX (instead of DW7) • GPGPU_DISPATCHDIMY (instead of DW8) • GPGPU_DISPATCHDIMZ (instead of DW9)
	9	**Workload Partition Enable** Format: Enable If set, the Workload Partition value in DW 13 is ignored and replaced by the current value of the WPARID MMIO register. **Programming Notes** The value of WPARID MMIO register must be a valid value for the Partition ID field in DW13.
	8	**Predicate Enable** Format: Enable If set, this command is executed (or not) depending on the current value of the MI Predicate internal state bit. This command is ignored only if **PredicateEnable** is set and the Predicate state bit is 0.
	7:0	**DWord Length** Format: =n Value: 37 — Name: Fixed Size
1	31:8	**Reserved**
	7:0	**Reserved** Access: RO Format: MBZ
2	31:30	**Partition Type** Format: U2 Specifies whether the command is executed by multiple partitions. When partitioned, the X or Y or Z dispatches are split at Partition Size boundaries.

COMPUTE_WALKER - COMPUTE_WALKER

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Disabled <b>[Default]</b></td><td>The command is not partitioned. Partition ID and Partition Size are ignored.</td></tr>
<tr><td>1</td><td>X</td><td>The command is partitioned in the X dimension. The X walk is between (PartitionID * PartitionSize) <= X < ((PartitionID+1)*PartitionSize). All Y and Z walks are performed in this partition.</td></tr>
<tr><td>2</td><td>Y</td><td>The command is partitioned in the Y dimension. The Y walk is between (PartitionID * PartitionSize) <= Y < ((PartitionID+1)*PartitionSize). All X and Z walks are performed in this partition.</td></tr>
<tr><td>3</td><td>Z</td><td>The command is partitioned in the Z dimension. The Z walk is between (PartitionID * PartitionSize) <= Z < ((PartitionID+1)*PartitionSize). All X and Y walks are performed in this partition.</td></tr>
</table>
</td></tr>
<tr><td></td><td>29:18</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>17</td><td><b>L3 prefetch disable</b>
<table>
<tr><td>Format:</td><td>Disable</td></tr>
</table>
If this bit is set, the prefetching of the indirect data to L3 is disabled.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This bit must not be set when INTERFACE_DESCRIPTOR for this command has BTD mode enabled for performance reasons.</td></tr>
</table>
</td></tr>
<tr><td></td><td>16:0</td><td><b>Indirect Data Length</b>
<table>
<tr><td>Format:</td><td>U17</td></tr>
</table>
This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored.
When present, the indirect data is pre-fetched into the L3 cache for the benefit of the threads that directly load their parameter data.
<table>
<tr><th>Restriction</th></tr>
<tr><td>Indirect Data Length is a multiple of 64 bytes (size of L3 cacheline). Bits [5:0] are zero.
Maximum supported value is 2^17(total GRF size * maximum threads/threadgroup). Typical value is much smaller:2^11 = 32 cache-lines.</td></tr>
</table>
</td></tr>
<tr><td>3</td><td>31:6</td><td><b>Indirect Data Start Address</b>
<table>
<tr><td>Format:</td><td>GeneralStateOffset[31:6]</td></tr>
</table>
This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the <b>General State Base Address.</b> It is the 64-byte aligned address of the indirect data.
The address is delivered to the kernel in the thread's R0 payload. The kernel is responsible for loading the indirect data from memory into the thread's registers for use.
</td></tr>
</table>

COMPUTE_WALKER - COMPUTE_WALKER

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>The thread payload layout is a kernel parameter convention coordinated between the driver that writes the indirect data, and the compiler prolog code that loads the indirect data into registers. Different API's may have different conventions.</td></tr>
<tr><td>This Indirect Data Start Address points the Global Constants data structure for Ray Tracing Shared function. This (address + 64B) points to the user defined Global Arguments for Ray Tracing. Unmodified address is sent to R0 as well as TraceRay Message header.</td></tr>
</table>
</td></tr>
<tr><td></td><td>5:0</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>4</td><td>31:30</td><td>
<b>SIMD Size</b>
This field determines the size of the payload and the number of bits of the execution mask that are expected. The kernel pointed to by the interface descriptor should match the SIMD declared here.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>SIMD8</td><td>8 LSBs of the execution mask are used</td></tr>
<tr><td>1</td><td>SIMD16</td><td>16 LSBs used in execution mask</td></tr>
<tr><td>2</td><td>SIMD32</td><td>32 bits of execution mask used</td></tr>
</table>
</td></tr>
<tr><td></td><td>29</td><td>
<b>Generate Local ID</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
If set, then Local ID will be generated by the thread dispatcher, using the Emit Local enable bits and the Local X/Y/Z Maximum values.
If clear, then auto-generated Local ID is disabled and Emit Local enable bits are ignored.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>If Bindless Thread dispatch (BTD mode) is enabled, then gen_local_id has to be disabled (i.e. 0)</td></tr>
</table>
<table>
<tr><th>Restriction</th></tr>
<tr><td>Auto-generation of Local ID requires at least 2 of the Local X/Y/Z maximum values to be a power of two, and that the Walk Order specify any dimension that is not a power of two as the lowest priority.
If more than 1 Local X/Y/Z maximum value is not a power of two, or if any Local dimension that is not a power of two is the first or second priority in the Walk Order, then Generate Local ID must not be set.</td></tr>
</table>
</td></tr>
<tr><td></td><td>28:26</td><td>
<b>Emit Local</b>
<table>
<tr><td>Format:</td><td>U3</td></tr>
</table>
These bits identify whether the register payload for Local X/Y/Z indices will be present. Bit 26 is X, Bit 27 is Y, and Bit 28 is Z. Separate <b>GPGPU_LOCALID</b> register
</td></tr>
</table>

COMPUTE_WALKER - COMPUTE_WALKER

payloads are generated when the corresponding bit is set.
If Generate Local ID is enabled, then the thread dispatcher generates the corresponding Local X/Y/Z index values, using the Local X/Y/Z Maximum values from DW6 of this command. For any enable bit that is not set, the corresponding Local ID will not be generated, and that register will not be emitted into the per-thread payload. When an enable bit is not set, its corresponding Local Maximum value in DW6 must be 0.

Value	Name
0	Emit None **[Default]**
1	Emit X
3	Emit XY
7	Emit XYZ
Others	Reserved

Programming Notes
For SIMD8 and SIMD16 threads, one GPGPU_LOCALID register is emitted to hold all the index values. If all X/Y/Z indices are present, then X is in R1, Y is in R2, and Z is in R3. For SIMD32 threads, a pair of GPGPU_LOCALID registers is emitted. The first register holds the lower 16 index values, and the second register holds the upper 16 index values. If all X/Y/Z indices are present, then X is in R1/R2, Y is in R3/R4, and Z is in R5/R6.
If Bindless Thread dispatch (BTD mode) is enabled, all the emit_local values must be set to EMIT_NONE

25

Emit Inline Parameter

Format:	Enable

When set, all threads in the threadgroup will have a payload register emitted with the Inline Data from this command (DW27..DW34). This register will immediately follow the register position for all the Local ID payloads. If all Emit Local bits are clear, this payload will be in R1.

Programming Notes
The Inline Parameter is used to pass up to 8 Dwords of addresses for kernel parameters. Passing them inline bypasses the memory latency of fetching those parameters from Indirect Data. See **GPGPU_INLINE_DATA** for the register layout.

24:22

Walk Order

Format:	U3

Specifies which dimensions are the first and second priority order for binding together in SIMD threads. In the values below, 0 is the first priority and 1 is the second priority.

COMPUTE_WALKER - COMPUTE_WALKER

Value	Name	Description
0	Walk 012 **[Default]**	Normal Linear walk order
1	Walk 021	
2	Walk 102	Normal TileY walk order
3	Walk 120	
4	Walk 201	
5	Walk 210	

21:19 **Tile Layout**

Format:	U3

Specifies whether 2D and 3D surfaces are stored in Linear or TileY layouts. The local ID values are batched together to keep full cache lines together in the same SIMD thread.

Value	Name
0	Linear **[Default]**
1	TileY 32bpe
2	TileY 64bpe
3	TileY 128bpe

18:17 **Message SIMD**

Format:	U2

Specifies the SIMD size of the messages used to access the local data. When the message size is less than the thread SIMD size, then the Local ID are batched so that the smaller message SIMD size keep full cache lines together in fused threads.

Value	Name
0	SIMD8
1	SIMD16
2	SIMD32

Restriction
Message SIMD must be <= Thread SIMD size.

16:1 **Reserved**

Access:	RO
Format:	MBZ

0 **Reserved**

Access:	RO
Format:	MBZ

<table>
<tr><th colspan="3">COMPUTE_WALKER - COMPUTE_WALKER</th></tr>
<tr><td>5</td><td>31:0</td><td><b>Execution Mask</b>
The execution mask is used with the last thread dispatched in a threadgroup, to mask off the SIMD lanes that are outside the range of number of local IDs in the group . All other threads dispatched in the threadgroup always have the all the SIMD lanes enabled.
All local IDs in the threadgroup are assumed to be fully packed into all the SIMD lanes, with only the last thread potentially having a partial SIMD lane use.
A SIMD32 thread uses all the execution mask bits. A SIMD16 thread uses the lower 16 bits of the execution mask. A SIMD8 thread uses the lower 8 bits of the execution mask.</td></tr>
<tr><td>6</td><td>31:30</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>29:20</td><td><b>Local Z Maximum</b>
<table>
<tr><td>Format:</td><td>U10</td></tr>
</table>
The maximum value of the threadgroup's Local ID in Z.
<table>
<tr><th>Restriction</th></tr>
<tr><td>The local workgroup size <b>LWS</b> =<b>(Local_X_Max+1)*(Local_Y_Max+1)*(Local_Z_Max+1).</b>
<b>LWS</b>must be <= 1024.
The number of dispatched threads <b>N</b> = <b>Number of Threads in GPGPU Thread Group</b> in the Interface Descriptor.
(<b>SIMTSize</b>) * (<b>N</b>-1) must be &lt; <b>LWS</b> &lt;= (<b>SIMTSize</b>) * <b>N</b>.</td></tr>
</table></td></tr>
<tr><td></td><td>19:10</td><td><b>Local Y Maximum</b>
<table>
<tr><td>Format:</td><td>U10</td></tr>
</table>
The maximum value of the threadgroup's Local ID in Y.</td></tr>
<tr><td></td><td>9:0</td><td><b>Local X Maximum</b>
<table>
<tr><td>Format:</td><td>U10</td></tr>
</table>
The maximum value of the threadgroup's Local ID in X.</td></tr>
<tr><td>7</td><td>31:0</td><td><b>Thread Group ID X Dimension</b>
The X dimension of the thread group (maximum X is dimension -1)
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>[1h, FFFFFFFFh]</td><td></td></tr>
</table></td></tr>
<tr><td>8</td><td>31:0</td><td><b>Thread Group ID Y Dimension</b>
The Y dimension of the thread group (maximum Y is dimension -1)
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>[1h, FFFFFFFFh]</td><td></td></tr>
</table></td></tr>
<tr><td>9</td><td>31:0</td><td><b>Thread Group ID Z Dimension</b>
The Z dimension of the thread group (maximum Z is dimension -1)</td></tr>
</table>

<table>
<tr><th colspan="3">COMPUTE_WALKER - COMPUTE_WALKER</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>[1h, FFFFFFFFh]</td><td></td></tr></table></td></tr>
<tr><td>10</td><td>31:0</td><td><b>Thread Group ID Starting X</b>
Specifies the initial value of the X component of the thread group when walker is started.
During the walker operation, when X is incremented to the <b>X Dimension</b> limit, on the next step itis re-loaded with the <b>Starting X</b> value.<table><tr><th>Restriction</th></tr><tr><td>When <b>Partition Type</b> is enabled, <b>Starting X</b> must be zero.</td></tr></table></td></tr>
<tr><td>11</td><td>31:0</td><td><b>Thread Group ID Starting Y</b>
Specifies the initial value of the Y component of the thread group when walker is started.
During the walker operation, when Y is incremented to the <b>Y Dimension</b> limit, on the next step itis re-loaded with the <b>Starting Y</b> value.<table><tr><th>Restriction</th></tr><tr><td>When <b>Partition Type</b> is enabled, <b>Starting Y</b> must be zero.</td></tr></table></td></tr>
<tr><td>12</td><td>31:0</td><td><b>Thread Group ID Starting Z</b>
Specifies the initial value of the Z component of the thread group when walker is started.
During the walker operation, when Z is incremented to the <b>Z Dimension</b> limit, on the next step itis re-loaded with the <b>Starting Z</b> value.<table><tr><th>Restriction</th></tr><tr><td>When <b>Partition Type</b> is enabled, <b>Starting Z</b> must be zero.</td></tr></table></td></tr>
<tr><td rowspan="2">13..14</td><td>63:32</td><td><b>Partition Size</b><table><tr><td>Format:</td><td>U32</td></tr></table>When this command's Partition Type is enabled, this specifies the size of the partition to use in the X/Y/Z direction.
When Partition Type is disabled, this field is ignored and the X/Y/Z dimensions are used.<table><tr><th>Programming Notes</th></tr><tr><td>When (Partition ID * Partition Size) > Dimension of the X/Y/Z direction, the walker terminates without making any dispatches.</td></tr><tr><td>The number of partitions NP =(Maximum Partition ID + 1). To dispatch the full walker range, (NP * Partition Size) must be >= Dimension of the X/Y/Z direction.</td></tr></table></td></tr>
<tr><td>31:0</td><td><b>Partition ID</b>
When this command's Partition Type is enabled, this specifies the partition number to use for this instance of the command. The Partition ID is in the range (0 .. number of partitions-1).</td></tr>
</table>

<table>
<tr><th colspan="3">COMPUTE_WALKER - COMPUTE_WALKER</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>[0-15]</td><td>Supported</td><td>A limited range of Partition ID values are supported by the COMPUTE_WALKER command.</td></tr>
<tr><td>Others</td><td>Reserved</td><td></td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Typical programming model sets this command's Indirect Partition Enable, so that the Partition ID is programmed from the command stream's WPARID MMIO register.
When not using partitioned execution, this field is set to zero.</td></tr>
</table>
</td></tr>
<tr><td>15</td><td>31:0</td><td><b>Preempt X</b>
Specifies the initial value of the X component of the thread group after walker is resumed (<b>CFE SubOpcode Variant</b> is Resume).
Must be zero when the walker is initially submitted (<b>CFE SubOpcode Variant</b> is Standard).</td></tr>
<tr><td>16</td><td>31:0</td><td><b>Preempt Y</b>
Specifies the initial value of the Y component of the thread group after walker is resumed (<b>CFE SubOpcode Variant</b> is Resume).
Must be zero when the walker is initially submitted (<b>CFE SubOpcode Variant</b> is Standard).</td></tr>
<tr><td>17</td><td>31:0</td><td><b>Preempt Z</b>
Specifies the initial value of the Z component of the thread group after walker is resumed (<b>CFE SubOpcode Variant</b> is Resume).
Must be zero when the walker is initially submitted (<b>CFE SubOpcode Variant</b> is Standard).</td></tr>
<tr><td>18..25
The Interface Descriptor describes the thread state common for all threads in the dispatch, including the Kernel base address, the binding tables, threadgroup size, and SLM size.</td><td>255:0</td><td><b>Interface Descriptor</b>
<table>
<tr><td>Format:</td><td>INTERFACE_DESCRIPTOR_DATA</td></tr>
</table>
</td></tr>
<tr><td>26..30
Post Sync command payload includes the operation, the address, a MOCS field, and an Immediate Data Value.</td><td>159:0</td><td><b>Post Sync</b>
<table>
<tr><td>Format:</td><td>POSTSYNC_DATA</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>When this command's Number of Partitions > 1, then there are that many instances of the Post Sync data structure located at the address in the POSTSYNC_DATA Address. (Number of Partitions = Max Partition ID + 1.)</td></tr>
</table>
</td></tr>
</table>

COMPUTE_WALKER - COMPUTE_WALKER		
		The Post Sync write operation is written to PostSync.Address + (Partition ID * datasize(PostSync.Operation)). The datasize varies based on Post Sync operation.
31..38	255:0	**Inline Data** Format: U32[8] When Emit Inline Parameter is enabled, this data is copied as the first cross-thread payload parameter for each thread.

Conditional Select

csel - Conditional Select

Source:	Eulsa
Length Bias:	4
Predication:	false
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

The csel instruction selectively moves components in src0 or src1 to the dst based on the result of the compare of src2 with zero. If the channel condition is true, data in src0 is moved into dst. Otherwise, data in src1 is moved into dst. The csel instruction provides the function of a cmp followed by sel. The instruction must not be used if cmpn is required. The instruction does not update the flag register.
The comparison follows the same rule as cmp instruction for that data type.

When Access Mode is Align1, accumulator may be used as source or destination.

Format:

```
csel (exec_size) dst src0 src1 src2
```

Syntax

```
csel[.cmod] (exec_size) reg reg reg reg
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     bitMask[n] = 0;
     if ( EMask.chan[n] ) {
         result[n] = src2.chan[n] - 0;
         bitMask[n] = Condition(result[n]);
         if (bitMask[n] = 1) {
             dst.chan[n] = src0.chan[n];
         } else {
             dst.chan[n] = src1.chan[n];
         }
     }
 }
```

Src Types	Dst Types
F	F
HF	HF
*D	*D
*W	*W

DWord	Bit	Description
0..3	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) AND ([Header][Opcode]!=madm) Format: **DirectOperand**

csel - Conditional Select		
	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) AND ([Header][Opcode]==madm) Format: **MacroOperand**
	127:112	**Src2.ImmValue[15:0]** Exists If: ([Src2.IsImm]==true)
	113:112	**Src2.HorzStride** Exists If: ([Src2.IsImm]==false) Format: **HorzStride**
	111:98	**Src1.Operand** Exists If: ([Header][Opcode]!=madm) Format: **DirectOperand**
	111:98	**Src1.Operand** Exists If: ([Header][Opcode]==madm) Format: **MacroOperand**
	97:96	**Src1.HorzStride** Format: **HorzStride**
	95:92	**CondCtrl** Format: **FlagModifier**
	91	**Src1.VertStride[1]** Format: **TernaryVertStride[1:1]**
	90:88	**Src1.DataType** Format: **TernaryDataType**
	87:86	**Src1.Mod** Format: **SrcMod**
	85:84	**Src2.Mod** Format: **SrcMod**
	83	**Src1.VertStride[0]** Format: **TernaryVertStride[0:0]**
	82:80	**Src2.DataType** Format: **TernaryDataType**
	79:66	**Src0.Operand** Exists If: ([Src0.IsImm]==false) AND ([Header][Opcode]!=madm) Format: **DirectOperand**
	79:66	**Src0.Operand** Exists If: ([Src0.IsImm]==false) AND ([Header][Opcode]==madm) Format: **MacroOperand**

csel - Conditional Select

<table>
<tr><td></td><td>79:64</td><td><b>Src0.ImmValue[15:0]</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>HorzStride</td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Header][Opcode]!=madm)</td></tr><tr><td>Format:</td><td>DirectOperand</td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Header][Opcode]==madm)</td></tr><tr><td>Format:</td><td>MacroOperand</td></tr></table></td></tr>
<tr><td></td><td>49</td><td><b>Reserved</b><table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>48</td><td><b>Dst.HorzStride</b>
This field provides the distance in unit of data elements between two adjacent data elements within a row (horizontal) in the register region for the operand.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>1 element</td></tr><tr><td>1</td><td>2 element</td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src2.IsImm</b>
This field indicate that Source 2 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td>SrcMod</td></tr></table></td></tr>
<tr><td></td><td>43</td><td><b>Src0.VertStride[1]</b><table><tr><td>Format:</td><td>TernaryVertStride[1:1]</td></tr></table></td></tr>
<tr><td></td><td>42:40</td><td><b>Src0.DataType</b><table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
<tr><td></td><td>39</td><td><b>ExecDataType</b>
This field indicate the datatype mode of ternary instruction. Integer or Float.</td></tr>
</table>

csel - Conditional Select

<table>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Integer</td></tr><tr><td>1</td><td>Float</td></tr></table></td></tr>
<tr><td></td><td>38:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>TernaryDataType</b></td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Src0.VertStride[0]</b><table><tr><td>Format:</td><td><b>TernaryVertStride[0:0]</b></td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td><b>Saturate</b></td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td><b>AccWrCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal
<b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction
<b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens</td></tr>
</table>

<table>
<tr><th colspan="3">csel - Conditional Select</th></tr>
<tr><td></td><td></td><td>after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b>
<table><tr><td>Format:</td><td><b>Header</b></td></tr></table></td></tr>
</table>

Constant Cache Dword Scattered Read MSD

MSD_CC_DWS - Constant Cache Dword Scattered Read MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message

MSD_CC_DWS - Constant Cache Dword Scattered Read MSD

	17:14	**Message Type** Default Value: 03h Format: Opcode Dword Scattered Read message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:10	**Reserved** Access: RO Format: MBZ
	9	**Legacy SIMD Mode** Default Value: 1h Format: Opcode Must be set for compatibility.
	8	**SIMD Mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Constant Cache Oword Aligned Block Read MSD

MSD_CC_OWAB - Constant Cache Oword Aligned Block Read MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Reserved** Access: RO Format: MBZ
	17:14	**Message Type** Default Value: 01h Format: Opcode Oword Aligned Block Read Constant Cache message
	13:11	**Reserved** Access: RO Format: MBZ
	10:8	**Data Elements** Format: MDC_DB_OW Specifies the number of contiguous Owords to be read
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Constant Cache Oword Block Read MSD

MSD_CC_OWB - Constant Cache Oword Block Read MSD

Source: EuSubFunctionReadOnlyDataPort

Length Bias: 1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Reserved** Access: RO Format: MBZ
	17:14	**Message Type** Default Value: 00h Format: Opcode Oword Block Read Constant Cache message
	13	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:11	**Reserved** Access: RO Format: MBZ
	10:8	**Data Elements** Format: MDC_DB_OW Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Continue

cont - Continue

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The cont instruction disables execution for the subset of channels for the remainder of the current loop iteration. Channels remain disabled until right before the while instruction or right before the condition check code block for the while instruction. If all enabled channels hit this instruction, jump to the instruction referenced by JIP where execution continues. UIP should always reference the loop's associated while instruction. JIP should point to the last instruction of the inner most conditional block if the cont instruction is inside a conditional block. In case of the break instruction directly under the loop, the JIP and the UIP are the same. If SPF is ON, the UIP must be used to update IP; JIP is not used in this case.

The following table describes the two 32-bit instruction pointer offsets. Both the JIP and UIP are signed 32-bit numbers, added to IP pre-increment. In instruction binary, JIP and UIP are at locations src0 and src1 and must be of type DW (signed DWord integer). When the offsets are immediate, src0 regfile must be immediate.

Format:

```
[(pred)] cont (exec_size) JIP UIP
```

Restriction
The execution size must be the same for the while, break, and cont instructions of the same code block.

Syntax

```
[(pred)] cont (exec_size) imm32 imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.channel[n] ) {
         if ( PMask[n] ) {  // PMask is for all channels enabled for the cont instruction.
             PcIP[n] = IP + UIP;
         } else {
             PcIP[n] = IP + 1;
         }
     }
 }
 for ( n = exec_size; n < 32; n++ ) {
     PcIP[n] = IP + 1;
 }
 if ( PcIP != (IP + 1) ) {  // all channels true
     Jump(IP + JIP);
 }
```

DWord	Bit	Description

cont - Continue

0..3	127:96	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	127:96	**JIP** Exists If: ([Src0.IsImm]==true) Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:80	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	95:64	**Reserved** Exists If: ([Src0.IsImm]==true) AND ([Src1.IsImm]==false) Format: MBZ
	95:64	**UIP** Exists If: ([Src0.IsImm]==true) AND ([Src1.IsImm]==true) Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	79:66	**Src0.Operand** Exists If: ([Src0.IsImm]==false) Format: DirectOperand
	65:64	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	63:50	**Dst.Operand** Format: DirectOperand
	49:48	**Reserved** Access: RO Format: MBZ
	47	**Src1.IsImm** This field indicate that Source 1 operand is carrying an immediate value Value | Name 0 | false 1 | true
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value Value | Name 0 | false

cont - Continue

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:34</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>BranchCtrl</b>
This field is used by <i>goto</i>, <b>if</b>, and <i>else</i> instructions to control branching. See the <b>goto</b> instruction description for more information about BranchCtrl.</td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr></table></td></tr>
</table>

cont - Continue

		1 | Negative | Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.
	27:24	**PredCtrl** Format: | **PredCtrl** This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.
	23	**FlagRegNum[0]** This field specifies bit[0] of the register number for a flag register operand.
	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: | **ChanOff** This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: | **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: | **Header**

Count Bits Set

cbit - Count Bits Set

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The cbit instruction counts component-wise the total bits set in src0 and stores the resulting counts in dst.

Format:

```
[(pred)] cbit (exec_size) dst src0
```

Restriction

No accumulator access, implicit or explicit.

Syntax

```
[(pred)] cbit (exec_size) reg reg
 [(pred)] cbit (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         UD cnt = 0;
         UD val = src0.chan[n];
         while ( val ) {
             if ( val & 1 ) {
                 cnt ++;
             }
             val = val » 1;
         }
         dst.chan[n] = cnt;
     }
 }
```

Src Types	Dst Types
UB, UW, UD	UD

DWord	Bit	Description
0..3	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true)
	95:92	**CondCtrl** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **FlagModifier**

cbit - Count Bits Set

95:64	**Src0.ImmValue[63:32]**		
		Exists If:	([Src0.IsImm]==true) AND (([Src0.DataType]==:q) OR ([Src0.DataType]==:uq) OR ([Src0.DataType]==:df))
87:84	**Src0.VertStride**		
		Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
		Format:	**VertStride**
83:81	**Src0.Width**		
		Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
		Format:	**Width**
80	**Src0.AddrMode**		
		Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
		Format:	**AddrMode**
79:66	**Src0.Operand**		
		Exists If:	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Direct)
		Format:	**DirectOperand**
79:66	**Src0.Operand**		
		Exists If:	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Indirect)
		Format:	**IndirectOperand**
65:64	**Src0.HorzStride**		
		Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
		Format:	**HorzStride**
63:50	**Dst.Operand**		
		Exists If:	([Dst.AddrMode]==Indirect)
		Format:	**IndirectOperand**
63:50	**Dst.Operand**		
		Exists If:	([Dst.AddrMode]==Direct)
		Format:	**DirectOperand**
49:48	**Dst.HorzStride**		
		Format:	**HorzStride**

cbit - Count Bits Set

47 **Reserved**

Access:	RO
Format:	MBZ

46 **Src0.IsImm**

This field indicate that Source 0 operand is carrying an immediate value.

Value	Name
0	false **[Default]**
1	true

45:44 **Src0.Mod**

Format:	**SrcMod**

43:40 **Src0.DataType**

Exists If:	([Src0.IsImm]==false)
Format:	**RegDataType**

43:40 **Src0.DataType**

Exists If:	([Src0.IsImm]==true)
Format:	**ImmDataType**

39:36 **Dst.DataType**

Format:	**RegDataType**

35 **Dst.AddrMode**

Format:	**AddrMode**

34 **Saturate**

Format:	**Saturate**

33 **AccWrCtrl**

Format:	**AccWrCtrl**

32 **AtomicCtrl**

Format:	**AtomicCtrl**

31 **MaskCtrl**

Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".

Value	Name	Description
0	Normal **[Default]**	Normal. Per channel write enable used for final write enable generation.
1	NoMask	NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.

30 **Reserved**

29 **CmptCtrl**

Format:	MBZ

<table>
<tr><th colspan="3">cbit - Count Bits Set</th></tr>
<tr><td></td><td></td><td>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td><b>PredCtrl</b></td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table>
<tr><td>Format:</td><td><b>ChanOff</b></td></tr>
</table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
</table>

cbit - Count Bits Set		
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Dot Product 4 Accumulate

dp4a - Dot Product 4 Accumulate

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	false

DP4A is a packed four-wide integer dot product and accumulate operation. Each source's 32-bit channel value is treated as four element vector of 8-bit integer values. The operation performs a 32-bit precision dot product of those four bytes and adds it with a 32-bit accumulator (typically a GRF, not necessarily an acc# reg).

Format:

```
[(pred)] dp4a (exec_size) dst src0 src1 src2
```

Programming Notes

```
EXAMPLE (SIMD1 for simplicity):
    mov  (1) r1.0:d  0x0102037F:d
    // (char4)(0x1,0x2,0x3,0x7F)
    mov  (1) r2.0:d  50:d
    dp4a (1) r3.0:d  r2:d r1:d r1:d
    // r3.0 = 50 + (0x1*0x1 + 0x2*0x2 + 0x3*0x3 + 0x7F*0x7F)
    //      = 50 + (1 + 4 + 9 + 16129)
    //      = 16193
```

Restriction

All three-source instructions have certain restrictions, described in Instruction Formats.

Only one of src0 or src1 operand may be the accumulator register (acc#).

Syntax

```
[(pred)] dp4a (exec_size) reg reg   reg reg
 [(pred)] dp4a (exec_size) reg imm16 reg reg
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst[n] = src0.chan[n]  +
             src1.chan[n][7:0]   * src2.chan[n][7:0] +
             src1.chan[n][15:8]  * src2.chan[n][15:8] +
             src1.chan[n][23:16] * src2.chan[n][23:16] +
             src1.chan[n][31:24] * src2.chan[n][31:24];
     }
 }
```

Src Types	Dst Types

dp4a - Dot Product 4 Accumulate

*D	*D

DWord	Bit	Description
0..3	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) AND ([Header][Opcode]!=madm) Format: **DirectOperand**
	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) AND ([Header][Opcode]==madm) Format: **MacroOperand**
	127:112	**Src2.ImmValue[15:0]** Exists If: ([Src2.IsImm]==true)
	113:112	**Src2.HorzStride** Exists If: ([Src2.IsImm]==false) Format: **HorzStride**
	111:98	**Src1.Operand** Exists If: ([Header][Opcode]!=madm) Format: **DirectOperand**
	111:98	**Src1.Operand** Exists If: ([Header][Opcode]==madm) Format: **MacroOperand**
	97:96	**Src1.HorzStride** Format: **HorzStride**
	95:92	**CondCtrl** Format: **FlagModifier**
	91	**Src1.VertStride[1]** Format: **TernaryVertStride[1:1]**
	90:88	**Src1.DataType** Format: **TernaryDataType**
	87:86	**Src1.Mod** Format: **SrcMod**
	85:84	**Src2.Mod** Format: **SrcMod**
	83	**Src1.VertStride[0]** Format: **TernaryVertStride[0:0]**
	82:80	**Src2.DataType** Format: **TernaryDataType**

dp4a - Dot Product 4 Accumulate

79:66 **Src0.Operand**

Exists If:	([Src0.IsImm]==false) AND ([Header][Opcode]!=madm)
Format:	DirectOperand

79:66 **Src0.Operand**

Exists If:	([Src0.IsImm]==false) AND ([Header][Opcode]==madm)
Format:	MacroOperand

79:64 **Src0.ImmValue[15:0]**

Exists If:	([Src0.IsImm]==true)

65:64 **Src0.HorzStride**

Exists If:	([Src0.IsImm]==false)
Format:	HorzStride

63:50 **Dst.Operand**

Exists If:	([Header][Opcode]!=madm)
Format:	DirectOperand

63:50 **Dst.Operand**

Exists If:	([Header][Opcode]==madm)
Format:	MacroOperand

49 **Reserved**

Format:	MBZ

48 **Dst.HorzStride**

This field provides the distance in unit of data elements between two adjacent data elements within a row (horizontal) in the register region for the operand.

Value	Name
0	1 element
1	2 element

47 **Src2.IsImm**

This field indicate that Source 2 operand is carrying an immediate value.

Value	Name
0	false
1	true

46 **Src0.IsImm**

This field indicate that Source 0 operand is carrying an immediate value.

Value	Name
0	false
1	true

45:44 **Src0.Mod**

Format:	SrcMod

<table>
<tr><th colspan="3">dp4a - Dot Product 4 Accumulate</th></tr>
<tr><td></td><td>43</td><td><b>Src0.VertStride[1]</b><table><tr><td>Format:</td><td><b>TernaryVertStride[1:1]</b></td></tr></table></td></tr>
<tr><td></td><td>42:40</td><td><b>Src0.DataType</b><table><tr><td>Format:</td><td><b>TernaryDataType</b></td></tr></table></td></tr>
<tr><td></td><td>39</td><td><b>ExecDataType</b>
This field indicate the datatype mode of ternary instruction. Integer or Float.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Integer</td></tr><tr><td>1</td><td>Float</td></tr></table></td></tr>
<tr><td></td><td>38:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>TernaryDataType</b></td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Src0.VertStride[0]</b><table><tr><td>Format:</td><td><b>TernaryVertStride[0:0]</b></td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td><b>Saturate</b></td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td><b>AccWrCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr></table></td></tr>
</table>

dp4a - Dot Product 4 Accumulate

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td>PredCtrl</td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td>ChanOff</td></tr></table>This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td>ExecSize</td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td>Header</td></tr></table></td></tr>
</table>

Dot Product Accumulate Systolic

dpas - Dot Product Accumulate Systolic

Source:	EuIsa
Length Bias:	4
Predication:	false
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false
Syntax:	GROUP
Subfunctions:	SystolicFC[49:48,45:43]

DPAS is a multiply add and accumulate operation of N elements in a systolic pipeline with low precision inputs (A x B). 32-bit SIMD channels are chunked into 32-bit/A and 32-bit/B elements (where A is the precision defined on Src1 and B is the precision of Src2). Example precisions are s2 for signed 2-bit, or u4 for unsigned 4-bit, bf for 16-bit bfloat16. The precision of the sources can vary per Src1 and Src2, but only certain combinations (described below) are permitted.

The Dst and Src0 take a regular type (e.g. :ud or :d or :f or :hf) and are treated as full 32/16-bit (Src0 as an accumulator to add operands to). The **sdepth** parameter is the systolic depth of the operation, meaning we perform a sequence of these operations advancing over successive registers. The output of each stage is a 32-bit value, which is the accumulated input to the next systolic stage. The first stage accumulation input is defined via the first source register (Src0). The last stage accumulated output is written to the destination register (Dst). The multiplier and the multiplicand come from Src1 and Src2 registers.

The **rcount** parameter is the RepeatCount of the operation, meaning rcount number of dpas instructions are generated with Dst and Src0 advancing successive registers, Src1 remaining same and Src2 advancing in units of Src2 datatype precision **sdepth** times the OPS_PER_CHAN (number of dot product operations per systolic channel).

A macro is defined as consecutive DPAS instructions of the same opcode, same datatype across all instructions, same register for Src1 and no producer-consumer relationships on it. Instructions in a macro can have variable repcounts. Macros must have the {Atomic} postfix used in all its instructions except for last one. See programming notes for examples of macro creation. Instructions produced through **rcount** are also considered as part of a macro.

All sources implicitly use <1;1,0> regioning, and the destination implicitly uses <1> regioning.

When Src0 is specified as null, it is treated as an immediate value of +0.

Format:

```
dpas.<sdepth>x<rcount> (exec_size) dst src0 src1 src2
```

Programming Notes

Example: Given any combination of datatypes in the sources of a DPAS instruction, the boundaries of a register should not be crossed.

i.e. in**dpas.8x8 (8) r24.0:ud r64.0:ud r4.0:u8 r44.0:u4**

Note that Src1 is of type U8 and Src2 is of type u4. Allowed subregisters in Src2 are r44.0 and r44.32

i.e. in**dpas.8x8 (8) r24.0:ud r64.0:ud r4.0:u4r44.0:u2**

Note that Src1 is of type U4and Src2 is of type u2. Allowed subregisters in Src2 are r44.0 and r44.64

i.e. in**dpas.8x8 (8) r24.0:ud r64.0:ud r4.0:u8 r44.0:u2**

dpas - Dot Product Accumulate Systolic

Note that Src1 is of type U8 and Src2 is of type u2. Allowed subregisters in Src2 are r44.0 r44.32, r44.64, and r44.96
i.e. in **dpas.8x8 (8) r24.0:ud r64.0:ud r4.0:u4r44.0:u4**
Note that Src1 is of type U4and Src2 is of type u4. Allowed subregisterin Src2 isr44.0

EXAMPLE 2: The GRF layout is independent of sdepth, so sdepth 8 can be emulated using 2 sdepth=4 instructions, also with subbyte datatype on src1 interleaving is used as shown below.

```
 dpas.4x2 (8) r24.0:d r64.0:d r4.0:u2 r14.0:u4 r24.0<1>:d =r64.0<1;1,0>:d +
   r4.0<16;8,1>:u2 . r14.0<0;8,1>:u4 +
   r4.8<16;8,1>:u2 . r14.8<0;8,1>:u4 +
   r5.0<16;8,1>:u2 . r14.16<0;8,1>:u4 +
   r5.8<16;8,1>:u2 . r14.24<0;8,1>:u4
 r25.0<1>:d =r65.0<1;1,0>:d +
   r4.0<16;8,1>:u2 . r15.0<0;8,1>:u4 +
   r4.8<16;8,1>:u2 . r15.8<0;8,1>:u4 +
   r5.0<16;8,1>:u2 . r15.16<0;8,1>:u4 +
   r5.8<16;8,1>:u2 . r15.24<0;8,1>:u4
 dpas.4x2 (8) r24.0:d r24.0:d r6.0:u2 r14.32:u4 r24.0<1>:d =r24.0<1;1,0>:d +
   r6.0<16;8,1>:u2 . r14.32<0;8,1>:u4 +
   r6.8<16;8,1>:u2 . r14.40<0;8,1>:u4 +
   r7.0<16;8,1>:u2 . r14.48<0;8,1>:u4 +
   r7.8<16;8,1>:u2 . r14.56<0;8,1>:u4
 r25.0<1>:d =r25.0<1;1,0>:d +
   r6.0<16;8,1>:u2 . r15.32<0;8,1>:u4 +
   r6.8<16;8,1>:u2 . r15.40<0;8,1>:u4 +
   r7.0<16;8,1>:u2 . r15.48<0;8,1>:u4 +
   r7.8<16;8,1>:u2 . r15.56<0;8,1>:u4
```

Example of a macro:

```
dpas.8x8 (8) r24.0:d r64.0:d r4.0:u2 r44.0:u4 {Atomic}
dpas.8x8 (8) r32.0:d r72.0:d r4.0:u2 r52.0:u4 {Atomic}
dpas.8x8 (8) r40.0:d r80.0:d r4.0:u2 r60.0:u4
```

Example of a macro:

```
dpas.8x8 (8) r24.0:d r64.0:d r4.0:u2 r44.0:u4 {Atomic}
dpas.8x4 (8) r32.0:d r72.0:d r4.0:u2 r52.0:u4 {Atomic}
dpas.8x2 (8) r40.0:d r80.0:d r4.0:u2 r60.0:u4
```

Restriction
All three-source instructions have certain restrictions, described in Instruction Formats.
Indirect address is not supported.
Dst and Src0 subregister offset is in units of its datatype precision and must be a multiple of ExecSize. Src1 subregister offsets must be 0.
Src2 subregister offset is in units of its datatype precision and must be a multiple of SystolicDepth times OPS_PER_CHAN.
General Accumulator registers access is not supported.
The Execution Size must be 8.
When instruction option **Atomic** is used it must be followed by a dpas instruction.
The systolic depth must be 8.

dpas - Dot Product Accumulate Systolic

The combinations of A (src1's precision) and B (src2's precision) supported are:

A(src1)	B(src2)	OPS_PER_CHAN
ub, b	ub, b	4
u4, s4, u2, s2	ub, b	4
ub, b	u4, s4, u2, s2	4
u4, s4, u2, s2	u4, s4, u2, s2	8

The combinations of Dst (destination's precision), Acc (src0'precision), A (src1's precision) and B (src2's precision) supported:

Dst	Acc(src0)	A(src1)	B(src2)	OPS_PER_CHAN
f	f	bf	bf	2

The combinations of Dst (destination's precision), Acc (src0'precision), A (src1's precision) and B (src2's precision) supported:

Dst	Acc(src0)	A(src1)	B(src2)	OPS_PER_CHAN
f	f	hf	hf	2

If any source datatype is signed, destination datatype must be signed

Bfloat denorms are always flushed to 0, and half-float denorm handling is based on the programmed denorm mode bit.
FP32/BF/HF final result is always Rounded to Nearest Even (RTNE).
FP32/BF Src0/Dest 's subnormal value always get flushed to zero.
HF Src0's subnormal value always get retained, but HF Dest's subnormal value always get flushed to zero.

Syntax

```
dpas.<sdepth>x<rcount> (exec_size) reg reg reg reg
```

Pseudocode

```
For Input and output are not DF datatype, Pseduo code as below:
 for (r = 0; r < rcount; r++) {
   // OPS_PER_CHAN is the number of dot product operations per systolic channel, V is Src2
in unit of Src2 datatype   V = Src2.InBits/Src2.DataTypePrecisionInBits + r * OPS_PER_CHAN
* 8;
   k = 0;
   // accumulated register input upconverted if required to internal accumulator precision
(32bit floats for float types)   temp = UpConvertToInternalPrecision( Src0.( RegNum + (r *
Src0.DataTypePrecisionInBits)/32 )(SubRegNum + (r * Src0.DataTypePrecisionInBits)%32) );
   for (s = 0; s < sdepth; s++) {
     Src1.OpsPerDword = 32 / (OPS_PER_CHAN * Src1.DataTypePrecisionInBits);
     U = Src1.( RegNum + (s » log2(Src1.OpsPerDword)) );
     for ( n = 0; n < exec_size; n++ ) {
       for ( d = 0; d < OPS_PER_CHAN; d++ ) {
         p = d + (s % Src1.OpsPerDword) * OPS_PER_CHAN;
         temp.chan[n] = temp.chan[n] + U.chan[n][p] * V[k+d];
       }
     }
     k += OPS_PER_CHAN;
   }
   // Write to output register, down converted to packed destination precision if required
```

dpas - Dot Product Accumulate Systolic

```
Dst.( RegNum + (r * Dst.DataTypePrecisionInBits)/32 )(SubRegNum + (r *
Dst.DataTypePrecisionInBits)%32)) = DownConvertToDstPrecision( temp );
 }
```

Src Types	Dst Types
*D,*B, U4, S4, U2, S2	*D
F, BF	F
F, HF	F

DWord	Bit	Description
0..3	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) Format: **DirectOperand**
	127:112	**Src2.ImmValue[15:0]** Exists If: ([Src2.IsImm]==true)
	113:112	**Reserved** Exists If: ([Src2.IsImm]==false) Format: MBZ
	111:98	**Src1.Operand** Format: **DirectOperand**
	97:96	**Reserved** Access: RO Format: MBZ
	95:92	**CondCtrl** Format: **FlagModifier**
	91	**Reserved** Access: RO Format: MBZ
	90:88	**Src1.DataType** Format: **TernaryDataType**
	87:86	**Src1.SubBytePrecision** Format: **SubBytePrecision**
	85:84	**Src2.SubBytePrecision** Format: **SubBytePrecision**
	83	**Reserved** Access: RO Format: MBZ

dpas - Dot Product Accumulate Systolic

82:80 **Src2.DataType**

Format:	**TernaryDataType**

79:66 **Src0.Operand**

Exists If:	([Src0.IsImm]==false)
Format:	**DirectOperand**

79:64 **Src0.ImmValue[15:0]**

Exists If:	([Src0.IsImm]==true)

65:64 **Reserved**

Exists If:	([Src0.IsImm]==false)
Format:	MBZ

63:50 **Dst.Operand**

Format:	**DirectOperand**

49:48 **SystolicDepth**

This field describes the systolic depth of the operation (the sdepth parameter in syntax).

Value	Name
0	16 deep
1	2 deep
2	4 deep
3	8 deep

47 **Src2.IsImm**

This field indicate that the src2 operand holds an immediate value.

Value	Name
0	false
1	true

46 **Src0.IsImm**

This field indicate that the src0 operand holds an immediate value.

Value	Name
0	false
1	true

45:43 **RepeatCount**

This field indicate the number of instructions to be created from a single macro instruction.

Value	Name
0	1
1	2
2	3
3	4
4	5

<table>
<tr><th colspan="3">dpas - Dot Product Accumulate Systolic</th></tr>
<tr><td></td><td></td><td><table><tr><td>5</td><td>6</td></tr><tr><td>6</td><td>7</td></tr><tr><td>7</td><td>8</td></tr></table></td></tr>
<tr><td></td><td>42:40</td><td><b>Src0.DataType</b><table><tr><td>Format:</td><td><b>TernaryDataType</b></td></tr></table></td></tr>
<tr><td></td><td>39</td><td><b>ExecDataType</b>
This field indicate the datatype mode of ternary instruction. Integer or Float.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Integer</td></tr><tr><td>1</td><td>Float</td></tr></table></td></tr>
<tr><td></td><td>38:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>TernaryDataType</b></td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Reserved</b><table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td><b>Saturate</b></td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td><b>AccWrCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.</td></tr>
</table>

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table>
</td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table>
</td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table><tr><td>Format:</td><td>PredCtrl</td></tr></table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table><tr><td>Format:</td><td>ChanOff</td></tr></table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table><tr><td>Format:</td><td>ExecSize</td></tr></table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b>
<table><tr><td>Format:</td><td>Header</td></tr></table>
</td></tr>
</table>

Dot Product Accumulate Systolic Wide

dpasw - Dot Product Accumulate Systolic Wide

Source:	EuIsa
Length Bias:	4
Predication:	false
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false
Syntax:	GROUP
Subfunctions:	SystolicFC[49:48,45:43]

DPAS wide is a multiply add and accumulate operation of N elements in a systolic pipeline with low precision inputs (A x B). 32-bit SIMD channels are chunked into 32-bit/A and 32-bit/B elements (where A is the precision defined on Src1 and B is the precision of Src2). Example precisions are s2 for signed 2-bit, or u4 for unsigned 4-bit, bf for 16-bit bfloat16. The precision of the sources can vary per Src1 and Src2, but only certain combinations (described below) are permitted.

The Dst and Src0 take a regular type (e.g. :ud or :d or :f or :hf) and are treated as full 32/16-bit (Src0 as an accumulator to add operands to). The **sdepth** parameter is the systolic depth of the operation, meaning we perform a sequence of these operations advancing over successive registers. The output of each stage is a 32-bit value, which is the accumulated input to the next systolic stage. The first stage accumulation input is defined via the first source register (Src0). The last stage accumulated output is written to the destination register (Dst). The multiplier and the multiplicand come from Src1 and Src2 registers.

The **rcount** parameter is the RepeatCount of the operation, meaning rcount number of dpas instructions are generated with Dst and Src0 advancing successive registers, Src1 remaining same and Src2 advancing in units of Src2 datatype precision **sdepth** times the OPS_PER_CHAN (number of dot product operations per systolic channel).

A macro is defined as consecutive DPAS instructions of the same opcode, same datatype across all instructions, same register for Src1 and no producer-consumer relationships on it. Instructions in a macro can have variable repcounts. Macros must have the {Atomic} postfix used in all its instructions except for last one. See programming notes for examples of macro creation. Instructions produced through **rcount** are also considered as part of a macro.

All sources implicitly use <1;1,0> regioning, and the destination implicitly uses <1> regioning.

When Src0 is specified as null, it is treated as an immediate value of +0.

DPAS wide differs from DPAS on that DPASw shares the data contents of the src2 register read from the GRF of one of the fused EUs among the two fused DPAS pipelines in a Fusion thread group. The rules on how this sharing is exercised are given below.

Format:

```
dpasw.<sdepth>x<rcount> (exec_size) dst src0 src1 src2
```

Programming Notes

For better performance, bank/bundles conflicts among sources that simultaneously read must be avoided by ensuring:

- Src0 and Src2 don't access the same bank

dpasw - Dot Product Accumulate Systolic Wide

- All sources dot access the same bundle

```
EXAMPLE:dpasw.8x2 (8) r24.0:d r64.0:d r4.0:ub r13.0:ubr24.0<1>:d =r64.0<1;1,0>:d +
r4.0<4;4,1>:ub . r13.0<0;4,1>:ub {read from EU0's GRF}+  r5.0<4;4,1>:ub . r13.4<0;4,1>:ub
{read from EU0's GRF}+  r6.0<4;4,1>:ub . r13.8<0;4,1>:ub {read from EU0's GRF}+
r7.0<4;4,1>:ub . r13.12<0;4,1>:ub {read from EU0's GRF}+  r8.0<4;4,1>:ub .
r13.16<0;4,1>:ub {read from EU0's GRF}+  r9.0<4;4,1>:ub . r13.20<0;4,1>:ub {read from
EU0's GRF}+  r10.0<4;4,1>:ub . r13.24<0;4,1>:ub {read from EU0's GRF}+  r11.0<4;4,1>:ub .
r13.28<0;4,1>:ub {read from EU0's GRF}r25.0<1>:d =r65.0<1;1,0>:d +  r4.0<4;4,1>:ub .
r13.0<0;4,1>:ub {read from EU1's GRF}+  r5.0<4;4,1>:ub . r13.4<0;4,1>:ub {read from EU1's
GRF}+  r6.0<4;4,1>:ub . r13.8<0;4,1>:ub {read from EU1's GRF}+  r7.0<4;4,1>:ub .
r13.12<0;4,1>:ub {read from EU1's GRF}+  r8.0<4;4,1>:ub . r13.16<0;4,1>:ub {read from
EU1's GRF}+  r9.0<4;4,1>:ub . r13.20<0;4,1>:ub {read from EU1's GRF}+  r10.0<4;4,1>:ub .
r13.24<0;4,1>:ub {read from EU1's GRF}+  r11.0<4;4,1>:ub . r13.28<0;4,1>:ub {read from
EU1's GRF}
```

The following table is an example with bytes in Src1 and Src2 showing which EU (EU0 or EU1) is providing the Src2 GRF for different iterations and RepeatCounts.

Repetition	DPAS sequence iteration	Src2 register	EU providing Src2 GRF
8	1	[Src2]	EU0
	2	[Src2]+1	EU0
	3	[Src2]+2	EU0
	4	[Src2]+3	EU0
	5	[Src2]	EU1
	6	[Src2]+1	EU1
	7	[Src2]+2	EU1
	8	[Src2]+3	EU1
7	1	[Src2]	EU0
	2	[Src2]+1	EU0
	3	[Src2]+2	EU0
	4	[Src2]+3	EU0
	5	[Src2]	EU1
	6	[Src2]+1	EU1
	7	[Src2]+2	EU1
6	1	[Src2]	EU0
	2	[Src2]+1	EU0

	3	[Src2]+2	EU0
	4	[Src2]	EU1
	5	[Src2]+1	EU1
	6	[Src2]+2	EU1
5	1	[Src2]	EU0
	2	[Src2]+1	EU0
	3	[Src2]+2	EU0
	4	[Src2]	EU1
	5	[Src2]+1	EU1
4	1	[Src2]	EU0
	2	[Src2]+1	EU0
	3	[Src2]	EU1
	4	[Src2]+1	EU1
3	1	[Src2]	EU0
	2	[Src2]+1	EU0
	3	[Src2]	EU1
2	1	[Src2]	EU0
	2	[Src2]	EU1
1	1	[Src2]	EU0

Exampleof a macro:

```
dpasw.8x8 (8) r24.0:d r64.0:d r4.0:u2 r44.0:u4 {Atomic}
dpasw.8x8 (8) r32.0:d r72.0:d r4.0:u2 r52.0:u4 {Atomic}
dpasw.8x8 (8) r40.0:d r80.0:d r4.0:u2 r60.0:u4
```

Exampleof a macro:

```
dpasw.8x8 (8) r24.0:d r64.0:d r4.0:u2 r44.0:u4 {Atomic}
dpasw.8x4 (8) r32.0:d r72.0:d r4.0:u2 r52.0:u4 {Atomic}
dpasw.8x2 (8) r40.0:d r80.0:d r4.0:u2 r60.0:u4
```

dpasw - Dot Product Accumulate Systolic Wide

Restriction
All three-source instructions have certain restrictions, described in Instruction Formats.
Indirect address is not supported.
Dst and Src0 subregister offset is in units of its datatype precision and must be a multiple of ExecSize. Src1 subregister offsets must be 0.
Src2 subregister offset is in units of its datatype precision and must be a multiple of SystolicDepth times OPS_PER_CHAN.
Accumulator registers access is not supported.
The Execution Size must be 8.
When instruction option **Atomic** is used it must be followed by a dpas instruction.
The systolic depth must be 8.
The combinations of A (src1's precision) and B (src2's precision) supported are:

A(src1)	B(src2)	OPS_PER_CHAN
ub, b	ub, b	4
u4, s4, u2, s2	ub, b	4
ub, b	u4, s4, u2, s2	4
u4, s4, u2, s2	u4, s4, u2, s2	8

The combinations of Dst (destination's precision), Acc (src0'precision), A (src1's precision) and B (src2's precision) supported:

Dst	Acc	A(src1)	B(src2)	OPS_PER_CHAN
f	f	bf	bf	2

The combinations of Dst (destination's precision), Acc (src0'precision), A (src1's precision) and B (src2's precision) supported:

Dst	Acc	A(src1)	B(src2)	OPS_PER_CHAN
f	f	hf	hf	2

Restriction
If any source datatype is signed, destination datatype must be signed
Bfloat denorms are always flushed to 0, and half-float denorm handling is based on the programmed denorm mode bit.
DPASw instructions should not be used in partial fused thread groups or in cases when code diverges. The behavior of DPASw when executed in these cases is undefined, and the user should not expect valid results from executing this case. Use DPAS in these cases instead.
DPASw source data must always start from channel 0.
Given any combination of datatypes in the sources of a DPAS instruction, the boundaries of a register should not be crossed. i.e. in**dpasw.8x8 (8) r24.0:ud r64.0:ud r4.0:u8 r44.0:u4** Note that Src1 is of type U8 and Src2 is of type u4. Allowed subregisters in Src2 are r44.0 and r44.32 i.e. in**dpasw.8x8 (8) r24.0:ud r64.0:ud r4.0:u4 r44.0:u2**

dpasw - Dot Product Accumulate Systolic Wide

Note that Src1 is of type U4and Src2 is of type u2. Allowed subregisters in Src2 are r44.0 and r44.64

i.e. in**dpasw.8x8 (8) r24.0:ud r64.0:ud r4.0:u8 r44.0:u2**

Note that Src1 is of type U8 and Src2 is of type u2. Allowed subregisters in Src2 are r44.0 r44.32, r44.64, and r44.96

i.e. in **dpasw.8x8 (8) r24.0:ud r64.0:ud r4.0:u4 r44.0:u4**

Note that Src1 is of type U4and Src2 is of type u4. Allowed subregisterin Src2 isr44.0

Syntax

```
dpasw.<sdepth>x<rcount> (exec_size) reg reg reg reg
```

Pseudocode

```
// OPS_PER_CHAN is the number of dot product operations per systolic channel
Src2.OpsPerDword = 32 / (OPS_PER_CHAN * Src2.DataTypePrecisionInBits);
 for (r = 0; r < rcount; r++) {
   // Src2 = EU0.Src2 denotes Src2 read from EU0's GRF. Src2 = EU1.Src2 means EU1's GRF
Src2 = (r < (Src2.OpsPerDword * ceiling( rcount /(2 * Src2.OpsPerDword )))) ? EU0.Src2 :
EU1.Src2;
   // V is in unit of Src2 datatype   V = Src2.InBits/Src2.DataTypePrecisionInBits + r *
OPS_PER_CHAN * 8;
   k = 0;
   // accumulated register input upconverted if required to internal accumulator precision
(32bit floats for float types)   temp = UpConvertToInternalPrecision( Src0.( RegNum + (r *
Src0.DataTypePrecisionInBits)/32 )(SubRegNum + (r * Src0.DataTypePrecisionInBits)%32) );
   for (s = 0; s < sdepth; s++) {
     Src1.OpsPerDword = 32 / (OPS_PER_CHAN * Src1.DataTypePrecisionInBits);
     U = Src1.( RegNum + (s » log2(Src1.OpsPerDword)) );
     for ( n = 0; n < exec_size; n++ ) {
       for ( d = 0; d < OPS_PER_CHAN; d++ ) {
         p = d + (s % Src1.OpsPerDword) * OPS_PER_CHAN;
         temp.chan[n] = temp.chan[n] + U.chan[n][p] * V[k][d];
       }
     }
     k += OPS_PER_CHAN;
   }
   // Write to output register, down converted to packed destination precision if required
Dst.( RegNum + (r * Dst.DataTypePrecisionInBits)/32 )(SubRegNum + (r *
Dst.DataTypePrecisionInBits)%32)) = DownConvertToDstPrecision( temp );
 }
```

Src Types	Dst Types
UD, D, UB, B, U4, S4, U2, S2	UD, D
F, BF	F
F, HF	F

<table>
<tr><th>DWord</th><th>Bit</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">0..3</td><td rowspan="3">127:114</td><td colspan="2"><b>Src2.Operand</b></td></tr>
<tr><td>Exists If:</td><td>([Src2.IsImm]==false)</td></tr>
<tr><td>Format:</td><td>DirectOperand</td></tr>
</table>

dpasw - Dot Product Accumulate Systolic Wide

127:112	**Src2.ImmValue[15:0]**		
		Exists If:	([Src2.IsImm]==true)
113:112	**Reserved**		
		Exists If:	([Src2.IsImm]==false)
		Format:	MBZ
111:98	**Src1.Operand**		
		Format:	**DirectOperand**
97:96	**Reserved**		
		Access:	RO
		Format:	MBZ
95:92	**CondCtrl**		
		Format:	**FlagModifier**
91	**Reserved**		
		Access:	RO
		Format:	MBZ
90:88	**Src1.DataType**		
		Format:	**TernaryDataType**
87:86	**Src1.SubBytePrecision**		
		Format:	**SubBytePrecision**
85:84	**Src2.SubBytePrecision**		
		Format:	**SubBytePrecision**
83	**Reserved**		
		Access:	RO
		Format:	MBZ
82:80	**Src2.DataType**		
		Format:	**TernaryDataType**
79:66	**Src0.Operand**		
		Exists If:	([Src0.IsImm]==false)
		Format:	**DirectOperand**
79:64	**Src0.ImmValue[15:0]**		
		Exists If:	([Src0.IsImm]==true)
65:64	**Reserved**		
		Exists If:	([Src0.IsImm]==false)
		Format:	MBZ
63:50	**Dst.Operand**		
		Format:	**DirectOperand**

<table>
<tr><th colspan="3">dpasw - Dot Product Accumulate Systolic Wide</th></tr>
<tr><td></td><td>49:48</td><td><b>SystolicDepth</b>
This field describes the systolic depth of the operation (the sdepth parameter in syntax).<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>16 deep</td></tr><tr><td>1</td><td>2 deep</td></tr><tr><td>2</td><td>4 deep</td></tr><tr><td>3</td><td>8 deep</td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src2.IsImm</b>
This field indicate that the src2 operand holds an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that the src0 operand holds an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:43</td><td><b>RepeatCount</b>
This field indicate the number of instructions to be created from a single macro instruction.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>2</td></tr><tr><td>2</td><td>3</td></tr><tr><td>3</td><td>4</td></tr><tr><td>4</td><td>5</td></tr><tr><td>5</td><td>6</td></tr><tr><td>6</td><td>7</td></tr><tr><td>7</td><td>8</td></tr></table></td></tr>
<tr><td></td><td>42:40</td><td><b>Src0.DataType</b><table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
<tr><td></td><td>39</td><td><b>ExecDataType</b>
This field indicate the datatype mode of ternary instruction. Integer or Float.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Integer</td></tr><tr><td>1</td><td>Float</td></tr></table></td></tr>
<tr><td></td><td>38:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">dpasw - Dot Product Accumulate Systolic Wide</th></tr>
<tr><td></td><td>35</td><td><b>Reserved</b><table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr></table></td></tr>
</table>

dpasw - Dot Product Accumulate Systolic Wide

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td><b>Header</b></td></tr></table></td></tr>
</table>

Dword Atomic Counter with Return Data Operation MSD

MSD1R_DWAC - Dword Atomic Counter with Return Data Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Bh Format: Opcode Atomic Counter Operation message

MSD1R_DWAC - Dword Atomic Counter with Return Data Operation MSD

	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2RS Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Dword Atomic Counter Write Only Operation MSD

MSD1W_DWAC - Dword Atomic Counter Write Only Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Bh Format: Opcode Atomic Counter Operation message

MSD1W_DWAC - Dword Atomic Counter Write Only Operation MSD

	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: **MDC_SM2RS** Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: **MDC_AOP** Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: **MDC_BTS** Specifies the Binding Table Index for the message

Dword Scattered Read MSD

MSD0R_DWS - Dword Scattered Read MSD

Source: EuSubFunctionDataPort0

Length Bias: 1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message

MSD0R_DWS - Dword Scattered Read MSD

<table>
<tr><td></td><td>17:14</td><td><b>Message Type</b><table><tr><td>Default Value:</td><td>03h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table>Dword Scattered Read message</td></tr>
<tr><td></td><td>13</td><td><b>Invalidate After Read</b><table><tr><td>Format:</td><td><b>MDC_IAR</b></td></tr></table>Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs</td></tr>
<tr><td></td><td>12:10</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>Legacy SIMD Mode</b><table><tr><td>Default Value:</td><td>1h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table>Must be set for compatibility.</td></tr>
<tr><td></td><td>8</td><td><b>SIMD Mode</b><table><tr><td>Format:</td><td><b>MDC_SM2</b></td></tr></table>Specifies the SIMD mode of the message (number of slots processed)</td></tr>
<tr><td></td><td>7:0</td><td><b>Binding Table Index</b><table><tr><td>Format:</td><td><b>MDC_BTS_A32</b></td></tr></table>Specifies the Binding Table Index for the message</td></tr>
</table>

Dword Scattered Write MSD

<table>
<tr><th colspan="3">MSD0W_DWS - Dword Scattered Write MSD</th></tr>
<tr><td colspan="3">Source: EuSubFunctionDataPort0
Length Bias: 1</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Packed Data Payload</b><table><tr><td>Default Value:</td><td>0 32 bit</td></tr><tr><td>Format:</td><td>Enable</td></tr></table>When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF).<table><tr><th>Restriction</th></tr><tr><td>Only 32-bit data packing is supported at this time.</td></tr></table></td></tr>
<tr><td></td><td>29</td><td><b>Packed Address Payload</b><table><tr><td>Default Value:</td><td>0 32 bit</td></tr><tr><td>Format:</td><td>Enable</td></tr></table>When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.</td></tr>
<tr><td></td><td>28:25</td><td><b>Message Length</b><table><tr><td>Format:</td><td>U4</td></tr></table>Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.</td></tr>
<tr><td></td><td>24:20</td><td><b>Response Length</b><table><tr><td>Format:</td><td>U5</td></tr></table>Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.</td></tr>
<tr><td></td><td>19</td><td><b>Header Present</b><table><tr><td>Format:</td><td>Enable</td></tr></table>If set, indicates that the message includes the header.</td></tr>
<tr><td></td><td>18</td><td><b>Legacy Message</b><table><tr><td>Default Value:</td><td>0h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table>Legacy Message</td></tr>
</table>

MSD0W_DWS - Dword Scattered Write MSD

	17:14	**Message Type** Default Value: 0Bh Format: Opcode Dword Scattered Write message
	13:10	**Reserved** Access: RO Format: MBZ
	9	**Legacy SIMD Mode** Default Value: 1h Format: Opcode Must be set for compatibility.
	8	**SIMD Mode** Format: **MDC_SM2** Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Binding Table Index** Format: **MDC_BTS_A32** Specifies the Binding Table Index for the message

Dword Typed Atomic Integer with Return Data Operation MSD

MSD1R_DWTAI - Dword Typed Atomic Integer with Return Data Operation MSD

Source: EuSubFunctionDataPort1

Length Bias: 1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Format: Opcode Typed Atomic Integer Operation message

MSD1R_DWTAI - Dword Typed Atomic Integer with Return Data Operation MSD

	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Slot Group** Format: **MDC_SG2** Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Format: **MDC_AOP** Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: **MDC_BTS** Specifies the Binding Table Index for the message

Dword Typed Atomic Integer Write Only Operation MSD

MSD1W_DWTAI - Dword Typed Atomic Integer Write Only Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Format: Opcode Typed Atomic Integer Operation message

MSD1W_DWTAI - Dword Typed Atomic Integer Write Only Operation MSD

	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Slot Group** Format: **MDC_SG2** Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Format: **MDC_AOP** Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: **MDC_BTS** Specifies the Binding Table Index for the message

Dword Untyped Atomic Float with Return Data Operation MSD

MSD1R_DWAF - Dword Untyped Atomic Float with Return Data Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 1Bh Format: Opcode Untyped Atomic Float Operation message

MSD1R_DWAF - Dword Untyped Atomic Float with Return Data Operation MSD

	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10:8	**Atomic Float Operation** Format: MDC_FOP Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword Untyped Atomic Float Write Only Operation MSD

MSD1W_DWAF - Dword Untyped Atomic Float Write Only Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 1Bh Format: Opcode Untyped Atomic Float Operation message

MSD1W_DWAF - Dword Untyped Atomic Float Write Only Operation MSD

	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11	**Data Width** Default Value: 0h Format: Opcode Operations are on 32-bit floats.
	10:8	**Atomic Float Operation** Format: MDC_FOP Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer with Return Data Operation MSD

MSD1R_DWAI - Dword Untyped Atomic Integer with Return Data Operation MSD

Source: EuSubFunctionDataPort1

Length Bias: 1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Format: Opcode Untyped Atomic Integer Operation message

MSD1R_DWAI - Dword Untyped Atomic Integer with Return Data Operation MSD

	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer Write Only Operation MSD

MSD1W_DWAI - Dword Untyped Atomic Integer Write Only Operation MSD

Source: EuSubFunctionDataPort1

Length Bias: 1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Format: Opcode Untyped Atomic Integer Operation message

MSD1W_DWAI - Dword Untyped Atomic Integer Write Only Operation MSD

	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: **MDC_SM2R** Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: **MDC_AOP** Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: **MDC_BTS_SLM_A32** Specifies the Binding Table Index for the message

Else

else - Else

Source:	EuIsa
Length Bias:	4
Predication:	false
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false
Syntax:	JUMP_BINARY_IMM_IMM

The else instruction is an optional statement within an if/else/endif block of code. It restricts execution within the else/endif portion to the opposite set of channels enabled under the if/else portion. Channels which were inactive prior to entering the if/endif block remain inactive throughout the entire block. All enabled channels upon arriving the else instruction will be redirected to the matching endif. If all channels are redirected (by else or before else), a relative jump is performed to the location specified by <JIP>. The jump target should be the matching endif instruction for that conditional block. The following table describes the 32-bit <JIP>. In instruction binary, <JIP> is at location <src1> and must be of type D (signed dword integer). <JIP> must be an immediate operand, it is a signed 32-bit number and is intended to be forward referencing. This value is added to IP pre-increment. If the <branch_ctrl> bit is set, then the <JIP> points to the first join instruction within the else block and <UIP> points to the endif instruction. If the <branch_ctrl> bit is not set, <JIP> and <UIP>, both point to endif.

Format:

```
        else (exec_size) JIP  UIP  branch_ctrl
```

Restriction
Predication is not allowed.
The execution size must be the same for the if, else, and endif instructions of the same code block.
The branch_ctrl must be set equal to (JIP != UIP).

Syntax

```
else (exec_size) imm32 imm32 branch_ctrl
```

Pseudocode

```
Evaluate(WrEn);
 for (n = 0; n < 32; n++) {
     if (WrEn.channel[n] == 1 || branch_ctrl) {
         PcIP[n] = IP + JIP;
     } else {
         PcIP[n] = IP + UIP;
     }
 }
 if (PcIP != (IP+1)) { // for all channels
     Jump(IP + JIP);
 }
```

DWord	Bit	Description

else - Else

0..3	127:96	**Reserved**	
		Exists If:	([Src0.IsImm]==false)
		Format:	MBZ
	127:96	**JIP**	
		Exists If:	([Src0.IsImm]==true)
		Format:	S31
		The byte-aligned jump distance if a jump is taken for the channel.	
	95:80	**Reserved**	
		Exists If:	([Src0.IsImm]==false)
		Format:	MBZ
	95:64	**Reserved**	
		Exists If:	([Src0.IsImm]==true) AND ([Src1.IsImm]==false)
		Format:	MBZ
	95:64	**UIP**	
		Exists If:	([Src0.IsImm]==true) AND ([Src1.IsImm]==true)
		Format:	S31
		The byte aligned jump distance if a jump is taken for the instruction.	
	79:66	**Src0.Operand**	
		Exists If:	([Src0.IsImm]==false)
		Format:	**DirectOperand**
	65:64	**Reserved**	
		Exists If:	([Src0.IsImm]==false)
		Format:	MBZ
	63:50	**Dst.Operand**	
		Format:	**DirectOperand**
	49:48	**Reserved**	
		Access:	RO
		Format:	MBZ
	47	**Src1.IsImm**	
		This field indicate that Source 1 operand is carrying an immediate value	
		Value	**Name**
		0	false
		1	true
	46	**Src0.IsImm**	
		This field indicate that Source 0 operand is carrying an immediate value	
		Value	**Name**
		0	false

<table>
<tr><th colspan="3">else - Else</th></tr>
<tr><td></td><td></td><td><table><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:34</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>BranchCtrl</b>
This field is used by <i>goto</i>, if, and <i>else</i> instructions to control branching. See the goto instruction description for more information about BranchCtrl.</td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr></table></td></tr>
</table>

else - Else

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td>PredCtrl</td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td>ChanOff</td></tr></table>This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td>ExecSize</td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td>Header</td></tr></table></td></tr>
</table>

End If

endif - End If

Source:	EuIsa
Length Bias:	4
Predication:	false
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The endif instruction terminates an if/else/endif block of code. It restores execution to the channels that were active prior to the if/else/endif block. The endif instruction is also used to hop out of nested conditionals by jumping to the end of the next outer conditional block when all channels are disabled.

The following table describes the 32-bit JIP. In instruction binary, JIP is at location src1 and must be of type D (signed DWord integer). JIP must be an immediate operand, it is a signed 32-bit number. This value is added to IP pre-increment.

Format:

```
        endif JIP
```

Restriction
Predication is not allowed.
The execution size must be the same for the if, else, and endif instructions of the same code block.

Syntax

```
endif (exec_size) imm32
```

Pseudocode

```
Evaluate(WrEn);
 if ( WrEn == 0 ) {  // all channels false
     Jump(IP + JIP);
 }
```

DWord	Bit	Description
0..3	127:96	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	127:96	**JIP** Exists If: ([Src0.IsImm]==true) Format: S31 The byte-aligned jump distance if a jump is taken for the channel
	95:80	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ

endif - End If

<table>
<tr><td></td><td>95:64</td><td><b>Reserved</b>
<table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b>
<table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Reserved</b>
<table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b>
<table><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:47</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:34</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>BranchCtrl</b>
This field is used by <b><i>goto</i></b>, <b>if</b>, and <b><i>else</i></b> instructions to control branching. See the <b>goto</b> instruction description for more information about BranchCtrl.</td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b>
<table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b>
<table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">endif - End If</th></tr>
<tr><td></td><td></td><td>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td>PredCtrl</td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table>
<tr><td>Format:</td><td>ChanOff</td></tr>
</table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
</table>

endif - End If

	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

EOT

MSD_EOT - EOT

Source:	EuSubFunctionGateway
Length Bias:	1

Specifies this is the last operation in this thread.

Restriction
The EOT bit in the EU SEND instruction must also be set when sending this message.
This message is only used with the following FFID thread dispatches: • COMPUTE_WALKER: FFID_GP, FFID_GP1 • FFID_MESH_SHADER, FFID_TASK_SHADER

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of GRF registers sent as the message payload. Value | Name | Description 1 | One **[Default]** | Data payload is ignored.
	24:20	**Response Length** Default Value: 0 None Format: U5 Specifies the number of GRF registers expected as the message response payload.
	19:16	**Reserved** Access: RO Format: MBZ
	15:12	**Fence Data Ports** Format: GW_FENCE_PORTS Bit mask specifies the list of data ports to be fenced with this EOT message. Restriction Ignored.
	11:6	**Reserved** Access: RO Format: MBZ

MSD_EOT - EOT

	5:4	**Reserved** Access: RO Format: MBZ
	3	**Reserved** Access: RO Format: MBZ
	2:0	**EOT Subfunction** Default Value: 0 Format: OpCode

Extended Math Function

math - Extended Math Function

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	true
Source Modifier:	true
Syntax:	GROUP
Subfunctions:	MathFC[95:92]

The math instruction performs extended math function on the components in src0, or src0 and src1, and write the output to the channels of dst. The type of extended math function are based on the FC[3:0] encoding in the table below.

Format:
[(pred)] math.<FC> (exec_size) dst src0
[(pred)] math.<FC> (exec_size) dst src0 src1

Restriction
Accumulator access is allowed only for IEEE macro functions (invm and rsqtm).
DF is only allowed for IEEE macro functions (invm and rsqtm).
The math instruction does not support indirect addressing modes.
The only supported rounding mode for math instruction is Round to Nearest Even.
INT DIV function does not support SIMD16. INT DIV function does not support simultaneous write to two registers. INT DIV function does not support source modifiers.
The FDIV function is not supported in ALT_MODE.
The math instruction can use GRF or immediates as source. The source formats for immediates must be one of the source formats supported by math operation.
DepCtrl must not be used.
The math instruction must use GRF as destination.
The supported regioning mode for math instruction is align1 and align16. The following restrictions apply for align1 mode: Scalar source is supported. Source and destination horizontal stride must be the same. Regioning must ensure Src.Vstride = Src.Width * Src.Hstride . Source and destination offset must be the same, except the case of scalar source.
Half-float denorms are always retained.
Math Operation rules when float and half-floats are mixed between source or between source and destination operands. The half-float operand must be interleaved in the register for align1 and the source and destination register offset must be the same to DW granularity. For align16, the half-float operand is allowed to be packed.
The execution size must be no more than 8 when half-floats are used in source or destination operand.
The source operand must not span two registers if the destination operand spans just one register Example:

math - Extended Math Function

Case (a) // Allowed. The source must be strided by 2. the offset is allowed to select between lower/upper 16b math.inv (8) r10:f r11.0<16;8,2>:hf math.inv (8) r10:f r11.1<16;8,2>:hf math.invm (8) r10:f r11.0<16;8,2>:hf r12.1<16;8,2>:hf Case (b) // Allowed. The destination must be strided by 2. The offset is allowed to selecte between lower/upper 16b math.inv (8) r10.0<2>:hf r11.0<8;8,1>:f math.inv (8) r10.1<2>:hf r11.0<8;8,1>:f math.invm (8) r10.0<2>:hf r11.0<16;8,2>:hf r12.0<16;8,2>:hf Case (c) // Allowed. Destination has stride of 2. The offset is allowed to select between uppoer/lower 16b math.invm (8) r10.0<2>:hf r11.0<8;8,1>:f r12.1<16;8,2>:hf math.invm (8) r10.1<2>:hf r11.1<16;8,2>:hf r12.0<8;8,1>:f Case (d) // Not Allowed. Destination is half-float but is not interleaved. math.inv (8) r10.0<1>:hf r11.0<8;8,1>:f Case (e) // Not Allowed. Source is half-float but not interleaved math.invm (8) r10.0<2>:hf r11.0<8;8,1>:f r12.0<8;8,1>:hf Case (f) // Not Allowed. Source operand spans 2 registers while destination spans one register. math.sin (8) r83.8<1>:hf r12.4<4;4,1>:f

Math Operation rules when half-floats are used on both source and destination operands. The execution size must be 8. The half-float source must be packed or interleaved. When interleaving, both source and destination must be interleaved. Example: Case (a) // Allowed. The source and destination are packed or interleaved math.inv (8) r10.0:hf r11.0<8;8,1>:hf math.inv (8) r10.0<2>:hf r11.0<16;8,2>:hf math.inv (8) r10.8:hf r11.0<8;8,1>:hf math.inv (8) r10.8<2>:hf r11.0<16;8,2>:hf

For one source math operations src1 must encode the null register.

Syntax

```
[(pred)] math.<FC> (exec_size) reg reg reg
```

Pseudocode

```
Evaluate(WrEn);
 for (n = 0; n < exec_size; n++) {
     if (WrEn.channel[n] == 1) {
         switch FC[3:0] {
             case 1h: // math.inv
         dst.channel[n] = rcp(src0.channel[n]);
             case 2h: // math.log
         dst.channel[n] = log(src0.channel[n]);
             case 3h: // math.exp
         dst.channel[n] = exp(src0.channel[n]);
             case 4h: // math.sqt
         dst.channel[n] = sqt(src0.channel[n]);
             case 5h: // math.rsqt
         dst.channel[n] = rsqt(src0.channel[n]);
             case 6h: // math.sin
         dst.channel[n] = sin(src0.channel[n]);
             case 7h: // math.cos
         dst.channel[n] = cos(src0.channel[n]);
             case 9h: // math.fdiv  (src0 / src1)
         dst.channel[n] = fdiv(src0.channel[n], src1.channel[n]);
             case Ah: // math.pow
         dst.channel[n] = pow(src0.channel[n], src1/channel[n]);
             case Bh: // math.idiv (src0 / src1)
         idiv(src0.channel[n], src1.channel[n]);
         dst.channel[n] = quotient;
         dst+1.channel[n] = remainder;
             case Ch: // math.iqot
         idiv(src0.channel[n], src1.channel[n]);
         dst.channel[n] = quotient;
```

math - Extended Math Function

```
            case Dh: // math.irem
        idiv(src0.channel[n], src1.channel[n]);
        dst.channel[n] = remainder;
            case Eh: // math.invm
        dst.channel[n] = invm(src0.channel[n], src1.channel[n]);
            case Fh: // math.rsqtm
        dst.channel[n] = rsqtm(src0.channel[n]);
            }
        }
    }
}
```

Src Types	Dst Types
F	F
HF	HF
DF	DF

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src0.IsImm]==false) AND ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true) AND ([Src1.IsImm]==false)
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src0.IsImm]==false) AND ([Src1.IsImm]==true)
	125:122	**Reserved** Exists If: ([Src0.IsImm]==false) AND ([Src1.IsImm]==false) Format: MBZ
	121:120	**Src1.Mod** Exists If: ([Src0.IsImm]==false) AND ([Src1.IsImm]==false) Format: **SrcMod**
	119:116	**Src1.VertStride** Exists If: ([Src0.IsImm]==false) AND ([Src1.IsImm]==false) Format: **VertStride**
	115:113	**Src1.Width** Exists If: ([Src0.IsImm]==false) AND ([Src1.IsImm]==false) Format: **Width**
	112	**Reserved** Exists If: ([Src0.IsImm]==false) AND ([Src1.IsImm]==false) Format: MBZ

math - Extended Math Function

111:98	**Src1.Operand**	
	Exists If:	([Src0.IsImm]==false) AND ([Src1.IsImm]==false) AND ([FuncCtrl]==INVM)
	Format:	**MacroOperand**
111:98	**Src1.Operand**	
	Exists If:	([Src0.IsImm]==false) AND ([Src1.IsImm]==false) AND ([FuncCtrl]!=INVM)
	Format:	**DirectOperand**
97:96	**Src1.HorzStride**	
	Exists If:	([Src0.IsImm]==false) AND ([Src1.IsImm]==false)
	Format:	**HorzStride**
95:92	**FuncCtrl**	
	Format:	**MathFC**
91:88	**Src1.DataType**	
	Exists If:	([Src0.IsImm]==false) AND ([Src1.IsImm]==true)
	Format:	**ImmDataType**
91:88	**Src1.DataType**	
	Exists If:	([Src0.IsImm]==false) AND ([Src1.IsImm]==false)
	Format:	**RegDataType**
91:64	**Reserved**	
	Exists If:	([Src0.IsImm]==true)
	Format:	MBZ
87:84	**Src0.VertStride**	
	Exists If:	([Src0.IsImm]==false)
	Format:	**VertStride**
83:81	**Src0.Width**	
	Exists If:	([Src0.IsImm]==false)
	Format:	**Width**
80	**Reserved**	
	Exists If:	([Src0.IsImm]==false)
	Format:	MBZ
79:66	**Src0.Operand**	
	Exists If:	([Src0.IsImm]==false) AND (([FuncCtrl]==INVM) OR ([FuncCtrl]==RSQTM))
	Format:	**MacroOperand**
79:66	**Src0.Operand**	
	Exists If:	([Src0.IsImm]==false) AND (([FuncCtrl]!=INVM) AND ([FuncCtrl]!=RSQTM))
	Format:	**DirectOperand**

<table>
<tr><th colspan="3">math - Extended Math Function</th></tr>
<tr><td></td><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([FuncCtrl]!=INVM) AND ([FuncCtrl]!=RSQTM)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([FuncCtrl]==INVM) OR ([FuncCtrl]==RSQTM)</td></tr><tr><td>Format:</td><td><b>MacroOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src1.IsImm</b>
This field indicate that Source 1 operand is carrying an immediate value<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Reserved</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">math - Extended Math Function</th></tr>
<tr><td></td><td>35</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr></table></td></tr>
</table>

math - Extended Math Function

		1 | Negative | Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.
	27:24	**PredCtrl** Format: **PredCtrl** This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.
	23	**FlagRegNum[0]** This field specifies bit[0] of the register number for a flag register operand.
	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: **ChanOff** This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Fence

DP_FENCE - Fence

Source:	SFID_1, SFID_6, SFID_D, SFID_E, SFID_F
Length Bias:	1

Wait until all previous accesses by this thread to this dataport are observable in the scope. Then optionally flush the cache.

Programming Notes
The src0 address payload specifies the flush address range.
The dest data payload specifies the notification status register (written to 0).

Restriction	Source
Fence is not supported by data port URB.	SFID_6

Syntax

```
[(pred)] FENCE[.scope].sfid[.flushtype] (exec_mask) dest_reg
<addr_type[+offset]>src0_reg:addr_size
```

Pseudocode

```
Wait_until_my_accesses_complete; dest.data = 0;
```

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>30:29</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>28:25</td><td><b>Src0 Length</b><table><tr><td>Format:</td><td>DP_ONE_ADDR_REG</td></tr></table>Specifies the size of the address payload, in registers.</td></tr>
<tr><td></td><td>24:20</td><td><b>Dest Length</b><table><tr><td>Format:</td><td>U5</td></tr></table>Specifies the size of destination data register payload.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>1</td><td></td><td>Completion signaled by write to register. No data returned.</td></tr></table></td></tr>
<tr><td></td><td>19</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

DP_FENCE - Fence

	18	**Reserved**
	17:16	**Reserved** Access: RO Format: MBZ
	15	**Flush Range** Default Value: 0 Full Cache Format: U1 Specifies if flush operation is the full cache or over an address range.
	14:12	**Flush Type** Format: **DP_FLUSH_TYPE** Specifies the type of cache flush operation to perform after the fence is complete. **Programming Notes** After fence completes, any flush operation is applied sequentially from the narrowest scope out to this scope level. This field is ignored if the FENCE message SFID is SLM.
	11:9	**Scope** Format: **DP_FENCE_SCOPE** Specifies the scope of the fence.
	8:7	**Reserved** Access: RO Format: MBZ
	6	**Reserved** Access: RO Format: MBZ
	5:0	**Operation** Default Value: 31 Fence Format: Opcode

Find First Bit from LSB Side

fbl - Find First Bit from LSB Side

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The fbl instruction counts component-wise the number of LSB 0 bits before the first 1 bit in src0, storing that number in dst.

Format:

```
        [(pred)] fbl (exec_size) dst src0
```

Programming Notes

If src0 contains no 1 bits, store 0xFFFFFFFF in dst.

Restriction

No accumulator access, implicit or explicit.

Syntax

```
[(pred)] fbl (exec_size) reg reg
 [(pred)] fbl (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         UD cnt = 0;
         UD udScalar = src0.chan[n];
         while ( (udScalar & 1) == 0 && cnt != 32 ) {
             cnt ++;
             udScalar = udScalar » 1;
         }
         if ( src0.chan[n] == 0x00000000 ) {
             dst.chan[n] = 0xFFFFFFFF;
         } else {
             dst.chan[n] = cnt;
         }
     }
 }
```

Src Types	Dst Types
UD	UD

DWord	Bit	Description

fbl - Find First Bit from LSB Side

0..3	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true)
	95:92	**CondCtrl** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **FlagModifier**
	95:64	**Src0.ImmValue[63:32]** Exists If: ([Src0.IsImm]==true) AND (([Src0.DataType]==:q) OR ([Src0.DataType]==:uq) OR ([Src0.DataType]==:df))
	87:84	**Src0.VertStride** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **VertStride**
	83:81	**Src0.Width** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **Width**
	80	**Src0.AddrMode** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **AddrMode**
	79:66	**Src0.Operand** Exists If: (([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Direct) Format: **DirectOperand**
	79:66	**Src0.Operand** Exists If: (([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Indirect) Format: **IndirectOperand**
	65:64	**Src0.HorzStride** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **HorzStride**
	63:50	**Dst.Operand** Exists If: ([Dst.AddrMode]==Indirect) Format: **IndirectOperand**

<table>
<tr><th colspan="3">fbl - Find First Bit from LSB Side</th></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td><b>Saturate</b></td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td><b>AccWrCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr></table></td></tr>
</table>

fbl - Find First Bit from LSB Side

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
 This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table> This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
 This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
 This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
</table>

fbl - Find First Bit from LSB Side

	21:19	**ChanOff** Format: **ChanOff** This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Find First Bit from MSB Side

fbh - Find First Bit from MSB Side

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

If src0 is unsigned, the fbh instruction counts component-wise the leading zeros from src0 and stores the resulting counts in dst. If src0 is signed and positive, the fbh instruction counts component-wise the leading zeros from src0 and stores the resulting counts in dst. If src0 is signed and negative, the fbh instruction counts component-wise the leading ones from src0 and stores the resulting counts in dst.

Format:

```
[(pred)] fbh (exec_size) dst src0
```

Programming Notes

If src0 is zero, store 0xFFFFFFFF in dst.

If src0 is signed and is -1 (0xFFFFFFFF), store 0xFFFFFFFF in dst.

Restriction

No accumulator access, implicit or explicit.

Syntax

```
[(pred)] fbh (exec_size) reg reg
 [(pred)] fbh (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         UD cnt = 0;
         if ( src0 is unsigned ) {
             UD udScalar = src0.chan[n];
             while ( (udScalar & (1 « 31)) == 0 && cnt != 32 ) {
                 cnt ++;
                 udScalar = udScalar « 1;
             }
             if ( src0.chan[n] == 0x00000000 ) {
                 dst.chan[n] = 0xFFFFFFFF;
             } else {
                 dst.chan[n] = cnt;
             }
         } else {  // src0 is signed.
             D dScalar = src0.chan[n];
             bit cval = dScalar[31];
             while ((dScalar & (1 « 31)) == cval && cnt != 32 ) {
                 cnt ++;
```

fbh - Find First Bit from MSB Side

```
                dScalar = dScalar « 1;
            }
            if ( (src0.chan[n] == 0xFFFFFFFF) || (src0.chan[n] == 0x00000000) ) {
                dst.chan[n] = 0xFFFFFFFF;
            } else {
                dst.chan[n] = cnt;
            }
        }
    }
}
```

Src Types	Dst Types
*D	UD

DWord	Bit	Description
0..3	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true)
	95:92	**CondCtrl** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **FlagModifier**
	95:64	**Src0.ImmValue[63:32]** Exists If: ([Src0.IsImm]==true) AND (([Src0.DataType]==:q) OR ([Src0.DataType]==:uq) OR ([Src0.DataType]==:df))
	87:84	**Src0.VertStride** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **VertStride**
	83:81	**Src0.Width** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **Width**
	80	**Src0.AddrMode** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **AddrMode**
	79:66	**Src0.Operand** Exists If: (([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Direct) Format: **DirectOperand**

fbh - Find First Bit from MSB Side

79:66	**Src0.Operand**	
	Exists If:	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Indirect)
	Format:	**IndirectOperand**
65:64	**Src0.HorzStride**	
	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
	Format:	**HorzStride**
63:50	**Dst.Operand**	
	Exists If:	([Dst.AddrMode]==Indirect)
	Format:	**IndirectOperand**
63:50	**Dst.Operand**	
	Exists If:	([Dst.AddrMode]==Direct)
	Format:	**DirectOperand**
49:48	**Dst.HorzStride**	
	Format:	**HorzStride**
47	**Reserved**	
	Access:	RO
	Format:	MBZ
46	**Src0.IsImm**	
	This field indicate that Source 0 operand is carrying an immediate value.	
	Value	**Name**
	0	false **[Default]**
	1	true
45:44	**Src0.Mod**	
	Format:	**SrcMod**
43:40	**Src0.DataType**	
	Exists If:	([Src0.IsImm]==false)
	Format:	**RegDataType**
43:40	**Src0.DataType**	
	Exists If:	([Src0.IsImm]==true)
	Format:	**ImmDataType**
39:36	**Dst.DataType**	
	Format:	**RegDataType**
35	**Dst.AddrMode**	
	Format:	**AddrMode**

fbh - Find First Bit from MSB Side

34 **Saturate**

Format:	Saturate

33 **AccWrCtrl**

Format:	AccWrCtrl

32 **AtomicCtrl**

Format:	AtomicCtrl

31 **MaskCtrl**

Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".

Value	Name	Description
0	Normal **[Default]**	Normal. Per channel write enable used for final write enable generation.
1	NoMask	NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.

30 **Reserved**

29 **CmptCtrl**

Format:	MBZ

Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.

Value	Name	Description
0	NoCompaction **[Default]**	No compaction. 128-bit native instruction supporting all instruction options.
1	Compacted	Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.

28 **PredInv**

This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields

Value	Name	Description
0	Positive **[Default]**	Positive polarity of predication. Use the predication mask produced by PredCtrl.
1	Negative	Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.

fbh - Find First Bit from MSB Side

	27:24	**PredCtrl** Format: **PredCtrl** This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.
	23	**FlagRegNum[0]** This field specifies bit[0] of the register number for a flag register operand.
	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: **ChanOff** This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Fraction

frc - Fraction

Source: Eulsa
Length Bias: 4
Predication: true
Conditional Modifier: true
Saturation: false
Source Modifier: true

The frc instruction computes, component-wise, the truncate-to-minus-infinity fractional values of src0 and stores the results in dst. The results, in the range of [0.0, 1.0], are the fractional portion of the source data. The result is in the range [0.0, 1.0] irrespective of the rounding mode. Floating-point fraction computation follows the rules in the following tables, based on the current floating-point mode.

Floating-Point Fraction Computation in IEEE Mode

src0	**-inf**	**-finite**	**-denorm**	**-0**	**+0**	**+denorm**	**+finite**	**+inf**	**NaN**
dst	NaN	*	+0/*^	+0	+0	+0/+denorm^	*	NaN	NaN
Notes:									
^	Result when denorm is enabled/supported.								
*	Result is in the range [+0.0, 1.0), not including 1.0.								

Floating-Point Fraction Computation in ALT Mode

src0	**-fmax**	**-finite**	**-denorm**	**-0**	**+0**	**+denorm**	**+finite**	**+fmax**	**
dst	+0	*	+0	+0	+0	+0	*	+0	
Notes:									
*	Result is in the range [+0.0, 1.0), not including 1.0.								
**	Result is +0 if src0 is {–inf, +inf, or NaN}.								

Format:

```
[(pred)] frc[.cmod] (exec_size) dst src0
```

Syntax

```
[(pred)] frc[.cmod] (exec_size) reg reg
 [(pred)] frc[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = src0.chan[n] - floor(src0.chan[n]);
     }
 }
```

Src Types	**Dst Types**
F	F

DWord	Bit	Description
0..3	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true)
	95:92	**CondCtrl** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **FlagModifier**
	95:64	**Src0.ImmValue[63:32]** Exists If: ([Src0.IsImm]==true) AND (([Src0.DataType]==:q) OR ([Src0.DataType]==:uq) OR ([Src0.DataType]==:df))
	87:84	**Src0.VertStride** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **VertStride**
	83:81	**Src0.Width** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **Width**
	80	**Src0.AddrMode** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **AddrMode**
	79:66	**Src0.Operand** Exists If: (([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Direct) Format: **DirectOperand**
	79:66	**Src0.Operand** Exists If: (([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Indirect) Format: **IndirectOperand**
	65:64	**Src0.HorzStride** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **HorzStride**
	63:50	**Dst.Operand** Exists If: ([Dst.AddrMode]==Indirect) Format: **IndirectOperand**

frc - Fraction

	63:50	**Dst.Operand** Exists If: ([Dst.AddrMode]==Direct) Format: **DirectOperand**
	49:48	**Dst.HorzStride** Format: **HorzStride**
	47	**Reserved** Access: RO Format: MBZ
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value. **Value** – **Name** 0 – false **[Default]** 1 – true
	45:44	**Src0.Mod** Format: **SrcMod**
	43:40	**Src0.DataType** Exists If: ([Src0.IsImm]==false) Format: **RegDataType**
	43:40	**Src0.DataType** Exists If: ([Src0.IsImm]==true) Format: **ImmDataType**
	39:36	**Dst.DataType** Format: **RegDataType**
	35	**Dst.AddrMode** Format: **AddrMode**
	34	**Saturate** Format: **Saturate**
	33	**AccWrCtrl** Format: **AccWrCtrl**
	32	**AtomicCtrl** Format: **AtomicCtrl**
	31	**MaskCtrl** Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0". **Value** – **Name** – **Description** 0 – Normal **[Default]** – Normal. Per channel write enable used for final write enable generation.

<table>
<tr><th colspan="3">frc - Fraction</th></tr>
<tr><td></td><td></td><td><table><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
</table>

frc - Fraction

	21:19	**ChanOff** Format: | **ChanOff** This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: | **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: | **Header**

Goto

goto - Goto

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The goto instruction directs the instruction pointer to the offset specified by the UIP offset or to the next IP based on the BranchCtrl bit in the instruction. When BranchCtrl is set the active channels that are predicated on this instruction will take IP + UIP path, the others will continue with IP + 1, the active channels that are not predicated on this instruction will be made inactive. Irrespective of BranchCtrl when there are no active channels the instruction pointer will move to IP + JIP.

The goto instruction is used in conjunction with a join instruction. A goto deactivates some channels that are reactivated at some program-specified join instruction. See the join instruction for the activation rules.

The goto and join instructions enable unstructured program control flow. These instructions must be used with additional care where dangling channels can result without proper compiler checks, meaning that it is expected that programs will navigate through these paths to reactivate the channels. Hardware does not provide native checks or reconvergence.

The following table describes the two 32-bit instruction pointer offsets. Both the JIP and UIP are signed 32-bit numbers, added to IP pre-increment. In instruction binary, JIP and UIP are at locations src0 and src1 and must be of type DW (signed DWord integer).

If SPF is ON, none of the PcIP are updated.

Format:

```
[(pred)] goto (exec_size) JIP UIP branch_ctrl
```

Restriction

Cannot have a goto in the body and the corresponding join in the function or the subroutine. Similarly the brd and brc.

Syntax

```
[(pred)] goto (exec_size) imm32 imm32 branch_ctrl
```

Pseudocode

```
Evaluate(WrEn);
for ( n = 0; n < exec_size; n++ ) {
    if ( WrEn.chan[n] ) { // for the predicated active channels
        PcIP[n] = IP + UIP;
    } else { // join IP, for the active non predicated channels
        PcIP[n] = IP + 1;
    }
}
if ( BranchCtrl ) {
    if (PcIP != (IP + UIP) ) { // for all channels
        if (PcIP != (IP + 1) ) { // for all channels
            Jump(IP + JIP);
        } else {
            Jump(IP + 1);
        }
    } else {
        Jump(IP + UIP);
    }
} else {
    if (PcIP != (IP + 1) ) { // for all channels
        Jump(IP + JIP);
    } else {
        Jump(IP + 1);
    }
}
```

DWord	Bit	Description

goto - Goto

0..3	127:96	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	127:96	**JIP** Exists If: ([Src0.IsImm]==true) Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:80	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	95:64	**Reserved** Exists If: ([Src0.IsImm]==true) AND ([Src1.IsImm]==false) Format: MBZ
	95:64	**UIP** Exists If: ([Src0.IsImm]==true) AND ([Src1.IsImm]==true) Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	79:66	**Src0.Operand** Exists If: ([Src0.IsImm]==false) Format: **DirectOperand**
	65:64	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	63:50	**Dst.Operand** Format: **DirectOperand**
	49:48	**Reserved** Access: RO Format: MBZ
	47	**Src1.IsImm** This field indicate that Source 1 operand is carrying an immediate value **Value** / **Name** 0 false 1 true
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value **Value** / **Name** 0 false

goto - Goto

1	true

45:34 **Reserved**

Access:	RO
Format:	MBZ

33 **BranchCtrl**

This field is used by ***goto***, **if,** and **else** instructions to control branching. See the **goto** instruction description for more information about BranchCtrl.

32 **AtomicCtrl**

Format:	**AtomicCtrl**

31 **MaskCtrl**

Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".

Value	Name	Description
0	Normal **[Default]**	Normal. Per channel write enable used for final write enable generation.
1	NoMask	NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.

30 **Reserved**

29 **CmptCtrl**

Format:	MBZ

Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.

Value	Name	Description
0	NoCompaction **[Default]**	No compaction. 128-bit native instruction supporting all instruction options.
1	Compacted	Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.

28 **PredInv**

This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields

Value	Name	Description
0	Positive **[Default]**	Positive polarity of predication. Use the predication mask produced by PredCtrl.

goto - Goto

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td><b>Header</b></td></tr></table></td></tr>
</table>

Half Precision HI8DS Render Target Write MSD

MSD_RTWH_HI8DS - Half Precision HI8DS Render Target Write MSD

Source:	EuSubFunctionRenderDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Message Precision Subtype** Default Value: 1h Format: Opcode Half precision data message
	29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Per-Coarse Pixel PS outputs enable** Format: Enable This bit indicates the render target write is a coarse pixel write. **Programming Notes** This bit must be DISABLED if a pixel shader thread is dispatched at pixel- or sample- rate. This bit may be DISABLED if a multi-rate pixel shader thread is dispatched at coarse rate. In such case, it indicates per-pixel or per-sample write (as determined by Per-Sample PS outputs enable) from a multi-rate shader, where the set of pixels is indicated by the Pixel shading phase field in Extended Message descriptor.

MSD_RTWH_HI8DS - Half Precision HI8DS Render Target Write MSD

<table>
<tr><td></td><td>17:14</td><td><b>Message Type</b><table><tr><td>Default Value:</td><td>0Ch</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table>Render Target Write message</td></tr>
<tr><td></td><td>13</td><td><b>Per-Sample PS Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot.<table><tr><th>Programming Notes</th></tr><tr><td>This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE.
When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0.
When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples).
When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.</td></tr></table></td></tr>
<tr><td></td><td>12</td><td><b>Last Render Target Select</b><table><tr><td>Format:</td><td>Enable</td></tr></table>This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.</td></tr>
<tr><td></td><td>11</td><td><b>Slot Group Select</b><table><tr><td>Format:</td><td>MDC_RT_SGS</td></tr></table>This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.</td></tr>
<tr><td></td><td>10:8</td><td><b>Render Target Message Subtype</b><table><tr><td>Default Value:</td><td>3h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table>SIMD8 dual source message. Use slots [15:8] for pixel enables, X/Y addresses, and oMask.<table><tr><th>Programming Notes</th></tr><tr><td>The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:24] are referenced instead of [15:8].</td></tr></table></td></tr>
</table>

MSD_RTWH_HI8DS - Half Precision HI8DS Render Target Write MSD

	7:0	**Binding Table Index** Format: **MDC_BTS** Specifies the Binding Table Index for the message

Half Precision LO8DS Render Target Write MSD

MSD_RTWH_LO8DS - Half Precision LO8DS Render Target Write MSD

Source:	EuSubFunctionRenderDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Message Precision Subtype** Default Value: 1h Format: Opcode Half precision data message
	29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Per-Coarse Pixel PS outputs enable** Format: Enable This bit indicates the render target write is a coarse pixel write. **Programming Notes** This bit must be DISABLED if a pixel shader thread is dispatched at pixel- or sample- rate. This bit may be DISABLED if a multi-rate pixel shader thread is dispatched at coarse rate. In such case, it indicates per-pixel or per-sample write (as determined by Per-Sample PS outputs enable) from a multi-rate shader, where the set of pixels is indicated by the Pixel shading phase field in Extended Message descriptor.

MSD_RTWH_LO8DS - Half Precision LO8DS Render Target Write MSD

	17:14	**Message Type** Default Value: 0Ch Format: Opcode Render Target Write message
	13	**Per-Sample PS Enable** Format: Enable If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot. **Programming Notes** This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Last Render Target Select** Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.
	11	**Slot Group Select** Format: MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
	10:8	**Render Target Message Subtype** Default Value: 2h Format: Opcode SIMD8 dual source message. Use slots [7:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].

MSD_RTWH_LO8DS - Half Precision LO8DS Render Target Write MSD

	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Half Precision REP16 Render Target Write MSD

MSD_RTWH_REP16 - Half Precision REP16 Render Target Write MSD

Source:	EuSubFunctionRenderDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Message Precision Subtype** Default Value: 1h Format: Opcode Half precision data message
	29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Per-Coarse Pixel PS outputs enable** Format: Enable This bit indicates the render target write is a coarse pixel write. **Programming Notes** This bit must be DISABLED if a pixel shader thread is dispatched at pixel- or sample- rate. This bit may be DISABLED if a multi-rate pixel shader thread is dispatched at coarse rate. In such case, it indicates per-pixel or per-sample write (as determined by Per-Sample PS outputs enable) from a multi-rate shader, where the set of pixels is indicated by the Pixel shading phase field in Extended Message descriptor.

MSD_RTWH_REP16 - Half Precision REP16 Render Target Write MSD

	17:14	**Message Type** Default Value: 0Ch Format: Opcode Render Target Write message
	13	**Per-Sample PS Enable** Format: Enable If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot. **Programming Notes** This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Last Render Target Select** Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.
	11	**Slot Group Select** Format: MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
	10:8	**Render Target Message Subtype** Default Value: 1h Format: Opcode SIMD16 Single source message with replicated data. Use slots [15:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].

MSD_RTWH_REP16 - Half Precision REP16 Render Target Write MSD

	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Half Precision SIMD8 Render Target Write MSD

MSD_RTWH_SIMD8 - Half Precision SIMD8 Render Target Write MSD

Source:	EuSubFunctionRenderDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Message Precision Subtype** Default Value: 1h Format: Opcode Half precision data message
	29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Per-Coarse Pixel PS outputs enable** Format: Enable This bit indicates the render target write is a coarse pixel write. **Programming Notes** This bit must be DISABLED if a pixel shader thread is dispatched at pixel- or sample- rate. This bit may be DISABLED if a multi-rate pixel shader thread is dispatched at coarse rate. In such case, it indicates per-pixel or per-sample write (as determined by Per-Sample PS outputs enable) from a multi-rate shader, where the set of pixels is indicated by the Pixel shading phase field in Extended Message descriptor.

MSD_RTWH_SIMD8 - Half Precision SIMD8 Render Target Write MSD

<table>
<tr><td></td><td>17:14</td><td><b>Message Type</b><table><tr><td>Default Value:</td><td>0Ch</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table>Render Target Write message</td></tr>
<tr><td></td><td>13</td><td><b>Per-Sample PS Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot.<table><tr><th>Programming Notes</th></tr><tr><td>This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE.
When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0.
When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples).
When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.</td></tr></table></td></tr>
<tr><td></td><td>12</td><td><b>Last Render Target Select</b><table><tr><td>Format:</td><td>Enable</td></tr></table>This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.</td></tr>
<tr><td></td><td>11</td><td><b>Slot Group Select</b><table><tr><td>Format:</td><td>MDC_RT_SGS</td></tr></table>This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.</td></tr>
<tr><td></td><td>10:8</td><td><b>Render Target Message Subtype</b><table><tr><td>Default Value:</td><td>4h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table>SIMD8 single source message. Use slots [7:0] for pixel enables, X/Y addresses, and oMask.<table><tr><th>Programming Notes</th></tr><tr><td>The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].</td></tr></table></td></tr>
</table>

	7:0	**Binding Table Index** Format: **MDC_BTS** Specifies the Binding Table Index for the message

Half Precision SIMD16 Render Target Write MSD

MSD_RTWH_SIMD16 - Half Precision SIMD16 Render Target Write MSD

Source:	EuSubFunctionRenderDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Message Precision Subtype** Default Value: 1h Format: Opcode Half precision data message
	29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Per-Coarse Pixel PS outputs enable** Format: Enable This bit indicates the render target write is a coarse pixel write. **Programming Notes** This bit must be DISABLED if a pixel shader thread is dispatched at pixel- or sample- rate. This bit may be DISABLED if a multi-rate pixel shader thread is dispatched at coarse rate. In such case, it indicates per-pixel or per-sample write (as determined by Per-Sample PS outputs enable) from a multi-rate shader, where the set of pixels is indicated by the Pixel shading phase field in Extended Message descriptor.

MSD_RTWH_SIMD16 - Half Precision SIMD16 Render Target Write MSD

	17:14	**Message Type** Default Value: 0Ch Format: Opcode Render Target Write message
	13	**Per-Sample PS Enable** Format: Enable If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot. **Programming Notes** This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Last Render Target Select** Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.
	11	**Slot Group Select** Format: MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
	10:8	**Render Target Message Subtype** Default Value: 0h Format: Opcode SIMD16 Single source message. Use slots [15:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].

MSD_RTWH_SIMD16 - Half Precision SIMD16 Render Target Write MSD		
	7:0	**Binding Table Index** Format: **MDC_BTS** Specifies the Binding Table Index for the message

Halt

halt - Halt

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The halt instruction temporarily suspends execution for all enabled compute channels. Upon execution, the enabled channels are sent to the instruction at (IP + UIP), if all channels are enabled at HALT, jump to the instruction at (IP + JIP). If the halt instruction is not inside any conditional code block, the values of JIP and UIP should be the same. If the halt instruction is inside a conditional code block, the UIP should be the end of the program and the JIP should be the end of the inner most conditional code block. The UIP must point to a HALT Instruction. If SPF is ON, the UIP must be used to update IP; JIP is not used in this case.

The following table describes the two 32-bit instruction pointer offsets. Both the JIP and UIP are signed 32-bit numbers, added to IP pre-increment. In instruction binary, JIP and UIP are at locations src0 and src1 and must be of type DW (signed DWord integer). When the offsets are immediate, src0 regfile must be immediate and dst must be null.

Format:

```
[(pred)] halt (exec_size) JIP UIP
```

Syntax

```
[(pred)] halt (exec_size) imm32 imm32
```

pre>

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < 32; n++ ) {
     if ( WrEn.channel[n] ) {
         PcIP[n] = IP + UIP;
     } else {
         PcIP[n] = IP + 1;
     }
 }
 if ( PcIP != (IP + 1) ) {  // for all channels
     Jump(IP + JIP);
 }
```

pre>

DWord	Bit	Description
0..3	127:96	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ

<table>
<tr><th colspan="3">halt - Halt</th></tr>
<tr><td></td><td>127:96</td><td><b>JIP</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td>S31</td></tr></table>The byte-aligned jump distance if a jump is taken for the channel.</td></tr>
<tr><td></td><td>95:80</td><td><b>Reserved</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>95:64</td><td><b>Reserved</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true) AND ([Src1.IsImm]==false)</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>95:64</td><td><b>UIP</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true) AND ([Src1.IsImm]==true)</td></tr><tr><td>Format:</td><td>S31</td></tr></table>The byte aligned jump distance if a jump is taken for the instruction.</td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>DirectOperand</td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Reserved</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Format:</td><td>DirectOperand</td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src1.IsImm</b>
This field indicate that Source 1 operand is carrying an immediate value<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">halt - Halt</th></tr>
<tr><td></td><td>45:34</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>BranchCtrl</b>
This field is used by <b><i>goto</i></b>, <b>if</b>, and <b><i>else</i></b> instructions to control branching. See the <b>goto</b> instruction description for more information about BranchCtrl.</td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
</table>

halt - Halt

	27:24	**PredCtrl** Format: **PredCtrl** This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.
	23	**FlagRegNum[0]** This field specifies bit[0] of the register number for a flag register operand.
	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: **ChanOff** This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

HCP_BSD_OBJECT

HCP_BSD_OBJECT		
Source:	VideoCS	
Length Bias:	2	
The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.		
The HCP_BSD_OBJECT command fetches the HEVC bit stream for a slice starting with the first byte in the slice. The bit stream ends with the last non-zero bit of the frame and does not include any zero-padding at the end of the bit stream. There can be multiple slices in a HEVC frame and thus this command can be issued multiple times per frame.		
The HCP_BSD_OBJECT command must be the last command issued in the sequence of batch commands before the HCP starts decoding. Prior to issuing this command, it is assumed that all configuration parameters in the HCP have been loaded including workload configuration registers and configuration tables. When this command is issued, the HCP is waiting for bit stream data to be presented to the shift register.		

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 20h HCP_BSD_OBJECT_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 1h — Name:

HCP_BSD_OBJECT		
1	31:0	**Indirect BSD Data Length** Format: U32 Specifies the length in bytes of the bitstream data for the current slice. It includes the first byte of the slice and the last non-zero byte of the in the slice. Specifically, the zero-padding bytes(if present) and the next start-code are excluded.
2	31:29	**Reserved** Access: RO Format: MBZ
	28:0	**Indirect Data Start Address** Format: U29 Specifies the byte-aligned graphics memory starting address of the slice bit stream relative to the **BSD Indirect Object Base Address**.

HCP_FQM_STATE

HCP_FQM_STATE

Source:	VideoCS
Length Bias:	2

The HCP_FQM_STATE command loads the custom HEVC quantization tables into local RAM and may be issued up to8 times: 4 scaling list per intra and inter.

Driver is responsible for performing the Scaling List division. So, save the division HW cost in HW. The 1/x value is provided in 16-bit fixed-point precision as ((1«17)/QM +1) » 1. .

Note: FQM is computed as (2^16)/QM. If QM=1, FQM=all 1's.

To simplify the design, only a limited number of scaling lists are provided at the PAK interface: default two SizeID0 and two SizeID123 (one set for inter and the other set for intra), and the encoder only allows custom entries for these four matrices. The DC value of SizeID2 and SizeID3 will be provided.

When the scaling_list_enable_flag is set to disable, the scaling matrix is still sent to the PAK, and with all entries programmed to the same value of 16.

This is a picture level state command and is issued in encoding processes only.

Dwords 2-33 form a table for the DCT coefficients, 2 16-bit coefficients/DWord.

- Size 4x4 for SizeID0, DWords 2-9.
- Size 8x8 for SizeID1/2/3, DWords 2-33.

SizeID 0 (Table 4-13)

4x4	[31:16]	[15:0]
DWord 2	AC(0,1)	DC
DWord 3	AC(0,3)	AC(0,2)
DWord 4	AC(1,1)	AC(1,0)
DWord 5	AC(1,3)	AC(1,2)
DWord 6	AC(2,1)	AC(2,0)
DWord 7	AC(2,3)	AC(2,2)
DWord 8	AC(3,1)	AC(3,0)
DWord 9	AC(3,3)	AC(3,2)

SizeID 1, 2, 3 (Table 4-14)

8x8	[31:16]	[15:0]
DWord 2	AC(0,1)	DC
DWord 3	AC(0,3)	AC(0,2)
DWord 4	AC(0,5)	AC(0,4)
DWord 5	AC(0,7)	AC(0,6)
DWord 6	AC(1,1)	AC(1,0)
DWord 7	AC(1,3)	AC(1,2)
DWord 8	AC(1,5)	AC(1,4)

HCP_FQM_STATE

DWord 9	AC(1,7)	AC(1,6)
...		
DWord 30	AC(7,1)	AC(7,0)
DWord 31	AC(7,3)	AC(7,2)
DWord 32	AC(7,5)	AC(7,4)
DWord 33	AC(7,7)	AC(7,6)

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline Type</b><table><tr><td>Default Value:</td><td>2h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:23</td><td><b>Media Instruction Opcode</b><table><tr><td>Default Value:</td><td>7h Codec/Engine Name</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table>Codec/Engine Name = HCP = 7h</td></tr>
<tr><td></td><td>22:16</td><td><b>Media Instruction Command</b><table><tr><td>Default Value:</td><td>5h HCP_FQM_STATE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>Dword Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>(Excludes Dwords 0, 1).<table><tr><th>Value</th><th>Name</th></tr><tr><td>20h</td><td></td></tr></table></td></tr>
<tr><td>1</td><td>31:16</td><td><b>FQM DC Value: (1/DC):</b><table><tr><td>Format:</td><td>U16</td></tr></table><table><tr><td>Specifies DC value of the scaling list for 16x16 (SizeID=2) or 32x32 (SizeID=3).</td></tr><tr><td>DC Value = scaling_list_dc_coef_minus8 + 8.</td></tr><tr><td>Driver will do the division.</td></tr></table></td></tr>
</table>

HCP_FQM_STATE

<table>
<tr><td></td><td>15:5</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>4:3</td><td><b>Color Component</b><table><tr><td>Format:</td><td>U2</td></tr></table>Luma and Chroma's share the same scaling list and DC value for the same SizeID.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Luma</td></tr><tr><td>1</td><td>Chroma Cb</td></tr><tr><td>2</td><td>Chroma Cr</td></tr><tr><td>3</td><td>Reserved</td></tr></table></td></tr>
<tr><td></td><td>2:1</td><td><b>SizeID</b><table><tr><td>Format:</td><td>U2</td></tr></table><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>SizeID 0 4x4</td></tr><tr><td>1</td><td>SizeID 1, 2, 3 (8x8, 16x16, 32x32)</td></tr><tr><td>2</td><td>SizeID 2 (for DC value in 16x16)</td></tr><tr><td>3</td><td>SizeID 3 (for DC value in 32x32)</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>Intra/Inter</b><table><tr><td>Format:</td><td>U1</td></tr></table>This field specifies the quant matrix intra or inter type.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Intra</td></tr><tr><td>1</td><td>Inter</td></tr></table></td></tr>
<tr><td>2..33</td><td>1023:0</td><td><b>QuantizerMatrix</b></td></tr>
</table>

HCP_IND_OBJ_BASE_ADDR_STATE

HCP_IND_OBJ_BASE_ADDR_STATE

Source:	VideoCS
Length Bias:	2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HCP_IND_OBJ_BASE_ADDR_STATE command is used to define the indirect object base address of the stream in graphics memory. This is a frame level command. (Is it frame or picture level?)
This is a picture level state command and is issued in both encoding and decoding processes.

Compressed Header Format

Fields	Bits	
Bin	0	Kernel Binarized Syntax
Probability select	1	0 ->indicates probability 128 1 ->indicates probability 256
	Repeat to pack a Cacheline	

Partition1 and TileSize record

Fields	Bits	
Tile Size	31:0	Partition1 Size is 16-bit value, Tile Size is 32-bit value
AddressOffset	63:32	Cacheline Address Offset to be Modified
Offset	69:64	Byte offset to be Modified
16-bit vs 32-bit update	70	0: Update 16-bit; 1: Update 32-bit
Reserved	511:71	

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 3h HCP_IND_OBJ_BASE_ADDR_STATE Format: OpCode

HCP_IND_OBJ_BASE_ADDR_STATE

	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1).
1..2	63:0	**HCP Indirect Bitstream Object Base Address** Format: SplitBaseAddress4KByteAligned Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the HCP_BSD_OBJECT command for fetching (reading) the compressed Slice Data.
3	31:0	**HCP Indirect Bitstream Object Memory Address Attributes** Format: MemoryAddressAttributes
4..5	63:0	**HCP Indirect Bitstream Object Access Upper Bound** Format: SplitBaseAddress4KByteAligned Decoder only. This field specifies the 4K-byte aligned maximum memory address access by the indirect data object in the HCP_BSD_OBJECT command for the slice bit stream. Indirect data accessed at this address or greater will cause the HCP to stop issuing requests to the GAC and the BSP VLD will then only receive zeros until a slice done is received. Setting this field to 0 will cause this range to be ignored by the HCP.
6..7	63:0	**HCP Indirect CU Object Base Address** Format: BaseAddress4KByteAligned Encoder only. Specifies the 4K-byte aligned data buffer base address for the read-only indirect data object for fetching (reading) per CU data during the encoding process.
8	31:0	**HCP Indirect CU Object Object Memory Address Attributes** Format: MemoryAddressAttributes
9..10	63:0	**HCP PAK-BSE Object Base Address** Format: BaseAddress4KByteAligned Encoder only. Specifies the 4K-byte aligned memory base address for the write-only data pointed by the PAK engine for writing out the compressed bitstream.
11	31:0	**HCP PAK-BSE Object Address Memory Address Attributes** Format: MemoryAddressAttributes Encoder only.

HCP_IND_OBJ_BASE_ADDR_STATE

12..13	63:0	**HCP PAK-BSE Object Access Upper Bound** Format: SplitBaseAddress4KByteAligned Encoder only. This field specifies the 4K-byte aligned maximum memory address access by the HCP_PAK_OBJECT command for writing out the slice bit stream. Indirect data accessed at this address and beyond will be blocked by the hardware and ignored. This address must be greater than the HCP PAK-BSE Object Base Address state.
14..15	63:0	**HCP VP9 PAK Compressed Header Syntax Streamin- Base Address** Exists If: //Encoder Only Format: BaseAddress4KByteAligned Specifies the 4K-byte aligned data buffer base address for the read-only Probability counters fetching during the encoding process..
16	31:0	**HCP VP9 PAK Compressed Header Syntax StreamIn Memory Address Attributes** Format: MemoryAddressAttributes
17..18	63:0	**HCP VP9 PAK Probability Counter StreamOut- Base Address** Exists If: //Encoder Only Format: BaseAddress4KByteAligned Specifies the 4K-byte aligned data buffer base address for the write-only Probability counters fetching during the encoding process.
19	31:0	**HCP VP9 PAK Probability Counter StreamOut Memory Address Attributes** Exists If: //Encoder Only Format: MemoryAddressAttributes
20..21	63:0	**HCP VP9 PAK Probability Deltas StreamIn- Base Address** Exists If: //Encoder Only Format: BaseAddress4KByteAligned Specifies the 4K-byte aligned data buffer base address for the read-only Probability differences during the encoding process.
22	31:0	**HCP VP9 PAK Probability Deltas StreamIn Memory Address Attributes** Exists If: //Encoder Only Format: MemoryAddressAttributes
23..24	63:0	**HCP VP9 PAK Tile Record StreamOut- Base Address** Format: BaseAddress4KByteAligned Specifies the 4K-byte aligned memory base address for the write-only data pointed by the PAK engine for writing out the Tile Record.
25	31:0	**HCP VP9 PAK Tile Record StreamOut Memory Address Attributes** Exists If: //Encoder Only Format: MemoryAddressAttributes

HCP_IND_OBJ_BASE_ADDR_STATE

26..27	63:0	**HCP VP9 PAK CU Level Statistic StreamOut- Base Address** Exists If: //Encoder Only Format: **BaseAddress4KByteAligned** Specifies the 4K-byte aligned memory base address for the write-only data pointed by the PAK engine for writing out the CU Record.
28	31:0	**HCP VP9 PAK CU Level Statistic StreamOut Memory Address Attributes** Exists If: //Encoder Only Format: **MemoryAddressAttributes**

HCP_PALETTE_INITIALIZER_STATE

HCP_PALETTE_INITIALIZER_STATE

Source:	VideoCS
Length Bias:	2

The HCP is selected with the **Media Instruction Opcode "7h"**for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HCP_PALETTE_INITIALIZER_STATE command loads in the SCC Palette Initializer Table to the HW.
Decoder only command.

Dword#2 - 193form a fixed size table for the Palette Initializer Table.
Max Palette Initializer Table is 128entries. Each entry has 3 components (Y, Cb and Cr) for a color.
Each component is 16-bits, even though currently only support up to 10-bit SCC extension. The upper (higher bits) 6 bits are set to zero - that is Least Significant Bit alignment.
Each entry of the Palette Initializer Table will consume 1.5 Dwords. Every two entries will consume 2 Dwords.
Hence, total requires 96 Dwords.
Dword#2 Bit 31 Cb#0 15:0 Luma#0 15:0 Bit 0
Dword#3 Bit 31 Luma#115:0 Cr#015:0 Bit 0
Dword#4 Bit 31 Cr#115:0 Cb#115:0 Bit 0
Dword#2 corresponds to the entry# 0 of the Palette Initializer Table.
Dword#193correspondsto the entry# 127of the Palette Initializer Table.

Programming Notes

Palette Initialization needs to happen at the beginning of each frame/tiles or start of each independent slice.
Palette initialization is not needed at the start of dependent slices (except the start of a new tiles since each tile needs to re-initialize the palette list) and the palette list is inherited from previous slice.
The following is the programming restriction:
(1) Palette Initialization command must be programmed in palette mode at the beginning of each frame and tiles (regardless if the slice is independent/dependent) and also the start of each independent slices.
(2) Palette Initialization command must not be programmed for dependent slices except the dependent slices are start of tiles (first slice in frame must be independent slice).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: Opcode
	28:27	**Pipeline Type** Default Value: 2h Format: Opcode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h

HCP_PALETTE_INITIALIZER_STATE

	22:16	**Media Instruction Command** Default Value: 9h HCP_PALETTE_INITIALIZER_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1) **Value** / **Name** C0h
1	31:8	**Reserved** Access: RO Format: MBZ
	7:0	**Active Palette Initializer Table Entries** The number of entries in the Palette Initializer Table that is valid. The Palette Initializer Table always filled with only valid entries starting from entry 0 onwards, packed and no jump. Max allowed 128entries. This field is set to 0 if there is no active color entry.
2..97	3071:0	**First 64 Color Entries** This contains first 64 color entries (0 to 63), each with 16-bit Y, U, V entries. DW2 = [31:0]; DW3 = [63:32], etc. for (i from 0 to 63) Palette Initializer Luma Value i = [(3 *i* 16 + 15) : (3 * i *16)] Palette Initializer Cb Value i = [((3 * i + 1) *16 + 15) : ((3 * i + 1) *16)] Palette Initializer CrValue i = [((3 * i + 2) *16 + 15) : ((3 * i + 2) *16)]
98..193	3071:0	**Second 64 Color Entries** This contains second64 color entries (64to 127), each with 16-bit Y, U, V entries. DW98= [31:0]; DW99= [63:32], etc. for (i from 64to 127) Palette Initializer Luma Value i = [(3 * (i-64) * 16 + 15) : (3 * (i-64)*16)] Palette Initializer Cb Value i = [((3 * (i-64) + 1) *16 + 15) : ((3 * (i-64)+ 1) *16)] Palette Initializer CrValue i = [((3 * (i-64) + 2) *16 + 15) : ((3 * (i-64) + 2) *16)]

HCP_PIPE_BUF_ADDR_STATE

<table>
<tr><th colspan="3">HCP_PIPE_BUF_ADDR_STATE</th></tr>
<tr><td colspan="3">Source: VideoCS
Length Bias: 2</td></tr>
<tr><td colspan="3">The HCP is selected with the <b>Media Instruction Opcode "7h"</b> for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.</td></tr>
<tr><td colspan="3">This state command provides the memory base addresses for the row store buffer and reconstructed picture output buffers required by the HCP.
This is a picture level state command and is shared by both encoding and decoding processes.</td></tr>
<tr><th colspan="3">Programming Notes</th></tr>
<tr><td colspan="3">All pixel surface addresses must be 4K byte aligned. There is a max of 8 Reference Picture Buffer Addresses, and all share the same third address DW in specifying 48-bit address.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline Type</b><table><tr><td>Default Value:</td><td>2h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:23</td><td><b>Media Instruction Opcode</b><table><tr><td>Default Value:</td><td>7h Codec/Engine Name</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table>Codec/Engine Name = HCP = 7h</td></tr>
<tr><td></td><td>22:16</td><td><b>Media Instruction Command</b><table><tr><td>Default Value:</td><td>2h HCP_PIPE_BUF_ADDR_STATE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>Dword Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>(Excludes Dwords 0, 1).<table><tr><th>Value</th><th>Name</th></tr><tr><td>66h</td><td></td></tr><tr><td>72h</td><td></td></tr></table></td></tr>
</table>

HCP_PIPE_BUF_ADDR_STATE

1..2	63:0	**Decoded Picture** Format: SplitBaseAddress4KByteAligned Frame buffer address for the final decoded picture YUV output.
3	31:0	**Decoded Picture Memory Address Attributes** Format: MemoryAddressAttributes
4..5	63:0	**Deblocking Filter Line Buffer** Format: SplitBaseAddress64ByteAligned Base address of the filter line buffer (read/write) used by the Deblocking Filter.
6	31:0	**Deblocking Filter Line Buffer Memory Address Attributes** Format: MemoryAddressAttributes
7..8	63:0	**Deblocking Filter Tile Line Buffer** Format: SplitBaseAddress64ByteAligned Base address of the tile line buffer (read/write) used by the Deblocking Filter.
9	31:0	**Deblocking Filter Tile Line Buffer Memory Address Attributes** Format: MemoryAddressAttributes
10..11	63:0	**Deblocking Filter Tile Column Buffer** Format: SplitBaseAddress64ByteAligned Base address of the tile column buffer (read/write) used by the Deblocking Filter.
12	31:0	**Deblocking Filter Tile Column Buffer Memory Address Attributes** Format: MemoryAddressAttributes
13..14	63:0	**Metadata Line Buffer** Format: SplitBaseAddress64ByteAligned Base address for the Metadata Line buffer.
15	31:0	**Metadata Line Buffer Memory Address Attributes** Format: MemoryAddressAttributes
16..17	63:0	**Metadata Tile Line Buffer** Format: SplitBaseAddress64ByteAligned Base address for the Metadata Tile Line buffer.
18	31:0	**Metadata Tile Line Buffer Memory Address Attributes** Format: MemoryAddressAttributes
19..20	63:0	**Metadata Tile Column Buffer** Format: SplitBaseAddress64ByteAligned Base address for the Metadata Tile Column buffer.
21	31:0	**Metadata Tile Column Buffer Memory Address Attributes** Format: MemoryAddressAttributes
22..23	63:0	**SAO Line Buffer** Format: SplitBaseAddress64ByteAligned Base address for the SAO Line buffer.

HCP_PIPE_BUF_ADDR_STATE		
24	31:0	**SAO Line Buffer Memory Address Attributes** Format: **MemoryAddressAttributes**
25..26	63:0	**SAO Tile Line Buffer** Format: **SplitBaseAddress64ByteAligned** Base address for the SAO Tile Line buffer.
27	31:0	**SAO Tile Line Buffer Memory Address Attributes** Format: **MemoryAddressAttributes**
28..29	63:0	**SAO Tile Column Buffer** Format: **SplitBaseAddress64ByteAligned** Base address for the SAO Tile Column buffer.
30	31:0	**SAO Tile Column Buffer Memory Address Attributes** Format: **MemoryAddressAttributes**
31..32	63:0	**Current Motion Vector Temporal Buffer** Format: **SplitBaseAddress64ByteAligned** Base address for the Current Motion Vector Temporal buffer.
33	31:0	**Current Motion Vector Temporal Buffer Memory Address Attributes** Format: **MemoryAddressAttributes**
34..35	63:0	**Reserved** Access: RO Format: MBZ
36	31:0	**Reserved** Access: RO Format: MBZ
37..52	511:0	**Reference Picture Base Address (RefAddr[0-7])** Format: **SplitBaseAddress64ByteAligned[8]** Base address of the reference picture buffer.
53	31:0	**Reference Picture Base Address Memory Address Attributes** Format: **MemoryAddressAttributes** **Programming Notes** Reference Picture Memory Compression Enables and Types (Media/Render) are moved to HCP_SURFACE_STATE. Separate8 Memory Compression Enables and Types are added in HCP_SURFACE_STATE so each reference picture surfaces have its own separate bits. The memory compression enable and type bit in this DW are not used.
54..55	63:0	**Original Uncompressed Picture Source** Format: **SplitBaseAddress64ByteAligned** Buffer address for fetching YUV pixel data from the original uncompressed input picture for

HCP_PIPE_BUF_ADDR_STATE

		encoding. This value is only valid in **encoding** mode.
56	31:0	**Original Uncompressed Picture Source Memory Address Attributes** Format: **MemoryAddressAttributes**
57..58	63:0	**Streamout Data Destination** Exists If: //Decoder Only Format: **SplitBaseAddress64ByteAligned** **Description** Buffer address for outputting the per-block indirect data to memory when **StreamOutEnable** is set in the HCP_PIPE_MODE_SELECT command. Decoder does not use this buffer. For Encoder: this field is used for dynamic repeat of frame in PAK for Rate Control. Also used for feeding coding information back to the Host, Video Preprocessing Unit and ENC Unit. For Encoder: This surface is used to streamout CU records All data are written in fixed formats, and therefore all record sizes are known in the hardware. Hardware can calculate the offset into this base address for per-block data.
59	31:0	**Streamout Data Destination Memory Address Attributes** Exists If: //Decoder Only Format: **MemoryAddressAttributes**
60..61	63:0	**Decoded Picture Status/Error Buffer Base Address or Encoded slice size streamout Base Address** Format: **SplitBaseAddress64ByteAligned** **Decoder Mode:** Specifies the 64 byte aligned buffer address for writing a single status/error cache-line sized record into memory when the Pic Status/Error Report Enable is set in the HCP_PIPE_MODE_SELECT command. The pic status/error record is written by hardware after the picture is decoded. **Encoder Mode:** This specifies 64 byte aligned buffer address for writing Slice size, when slice size conformance is enabled.
62	31:0	**Decoded Picture Status/Error Buffer Base Address Memory Address Attributes** Format: **MemoryAddressAttributes**
63..64	63:0	**LCU ILDB Streamout Buffer** Format: **SplitBaseAddress64ByteAligned** Buffer address for writing ILDB parameter per LCU to memory when Deblocker Streamout Enable is set in the HCP_PIPE_MODE_SELECT Command. The ILDB MB control parameters are written by HW at the end of each reconstructed LCU. Only edge information is being streamed out.

HCP_PIPE_BUF_ADDR_STATE		
65	31:0	**LCU ILDB Streamout Buffer Memory Address Attributes** Format: **MemoryAddressAttributes**
66..81	511:0	**Collocated Motion Vector Temporal Buffer[0-7]** Format: **SplitBaseAddress64ByteAligned[8]** Base address for the Collocated Motion Vector Temporal buffer.
82	31:0	**Collocated Motion Vector Temporal Buffer[0-7] Memory Address Attributes** Format: **MemoryAddressAttributes**
83..84	63:0	**VP9 Probability Buffer Read/Write** Format: **SplitBaseAddress64ByteAligned** Specifies the 64 byte aligned buffer address for VP9 Probability Buffer. Hardware reads in the probability for decode and write out the modified probability for future frames. Driver needs to program the Initial VP9 Probability for decoding the current frame. For Key Frame, it should contain the default Key Frame Probability. For non-Key Frame, it could be a default (non-Key) or one of the 8 Reference Buffers Probability. Driver must provide a valid Initial VP9 Probability buffer.
85	31:0	**VP9 Probability Buffer Read/Write Memory Address Attributes** Format: **MemoryAddressAttributes**
86..87	63:0	**VP9 Segment ID Buffer Read/Write** Specifies the 64 byte aligned buffer address for VP9 SegmentID buffer. This should contain the writeout SegmentID from previous frame and will be used to predict SegmentID for the current frame. Hardware will write out SegmentID of the current frame in the same address for the next frame.
88	31:0	**VP9 Segment ID buffer Read/Write Memory Address Attributes** Format: **MemoryAddressAttributes**
89..90	63:0	**VP9 HVD Line Rowstore Buffer Read/Write** Format: **SplitBaseAddress64ByteAligned** Specifies the 64 byte aligned buffer address for HVD Tile Rowstore Buffer (bitstream decoder).
91	31:0	**VP9 HVD Line Rowstore buffer Read/Write Memory Address Attributes** Format: **MemoryAddressAttributes**
92..93	63:0	**VP9 HVD Tile Rowstore Buffer Read/Write** Format: **SplitBaseAddress64ByteAligned**
94	31:0	**VP9 HVD Tile Rowstore buffer Read/Write Memory Address Attributes** Format: **MemoryAddressAttributes**
95..96	63:0	**SAO Rowstore Buffer Base Address** Specifies the 64 byte aligned buffer address for Rowstoring of SAO parameters in encoder mode
97	31:0	**SAO Rowstore Buffer Read/Write Memory Address Attributes** Format: **MemoryAddressAttributes**

HCP_PIPE_BUF_ADDR_STATE		
98..99	63:0	**Frame Statistics Streamout Data Destination Buffer Base Address** Format: **SplitBaseAddress64ByteAligned** **Description** Specifies the 64 byte aligned buffer address for outputting the frame statistics data to memory. The statistics are mainly SliceSize conformance, SSE, RhoDomain and CU parameters. Decoder does not use this buffer.
100	31:0	**Frame Statistics Streamout Data Destination buffer (attributes) Read/Write** Format: **MemoryAddressAttributes**
101..102	63:0	**SSE Source Pixel RowStore Buffer Base Address** Format: **SplitBaseAddress64ByteAligned** Specifies the 64 byte aligned buffer address for storing the source pixels for SSE. SSE metrics in the PAK are computed using post loop-filtered pixels or post SAO, if SAO is enabled.
103	31:0	**SSE Source Pixel RowStore buffer (attributes) Read/Write** Format: **MemoryAddressAttributes**
104..105	63:0	**HCP Scalability Slice State Buffer Base Address** Format: **SplitBaseAddress64ByteAligned** Specifies the 64 byte aligned buffer address for storing slice state information on HEVC/VP9 Scalable mode for decode. This is needed since CABAC and BE pass will be separated and BE pass needs to have slice state information as well. This buffer is only used in HEVC Scalable Decode Mode Only (Virtual Tile on both CABAC and Recon Pass)
106	31:0	**HCP Scalability Slice State Buffer (attributes) Read/Write** Format: **MemoryAddressAttributes**
107..108	63:0	**HCP Scalability CABAC Decoded Syntax Elements Buffer Base Address** Format: **SplitBaseAddress64ByteAligned** Specifies the 64 byte aligned buffer address for storing CABAC Decoded Syntax Element on HEVC/VP9 Scalable mode for decode
109	31:0	**HCP Scalability CABAC Decoded Syntax Elements Buffer (attributes) Read/Write** Format: **MemoryAddressAttributes**
110..111	63:0	**Motion Vector Upper Right Column Store Buffer Base Address** Format: **SplitBaseAddress64ByteAligned** Specifies the 64 byte aligned buffer address for storing upper right Motion Vector on HEVC /VP9 Scalable mode for decode in BE pass. This buffer is used to pass data across pipes for multiple pipe mode. This buffer is only used on HEVC Scalable Decode Only (Virtual Tile on both CABAC and Recon pass)
112	31:0	**Motion Vector Upper Right Column Store Buffer (attributes) Read/Write** Format: **MemoryAddressAttributes**

HCP_PIPE_BUF_ADDR_STATE		
113..114	63:0	**Intra Prediction Upper Right Column Store Buffer Base Address** Format: SplitBaseAddress64ByteAligned Specifies the 64 byte aligned buffer address for storing upper right Intra Prediction Pixel on HEVC /VP9 Scalable mode for decode in BE pass. This buffer is used to pass data across pipes for multiple pipe mode.
115	31:0	**Intra Prediction Upper Right Column Store Buffer (attributes) Read/Write** Format: MemoryAddressAttributes
116..117	63:0	**Intra Prediction Left Recon Column Store Buffer Base Address** Format: SplitBaseAddress64ByteAligned Specifies the 64 byte aligned buffer address for storing left column Intra Prediction Pixel on HEVC /VP9 Scalable mode for decode in BE pass. This buffer is used to pass data across pipes for multiple pipe mode.
118	31:0	**Intra Prediction Left Recon Column Store Buffer (attributes) Read/Write** Format: MemoryAddressAttributes
119..120	63:0	**HCP Scalability CABAC Decoded Syntax Elements Buffer Max Address** Format: SplitBaseAddress64ByteAligned Specifies the 64 byte aligned maximum address for the HCP Scalability CABAC Decoded Syntax Elements Buffer. This address shall either be 0 or larger than HCP Scalability CABAC Decoded Syntax Elements Buffer Base Address. If this address is 0, the upper bound is considered disable and HW will NOT check for upper bound. Hardware shall only write to memory address less than this address (unless address is 0 which is disabled). Hardware will not write to memory address larger than or equal to address (unless address is 0 which is disabled)

HCP_PIPE_MODE_SELECT

HCP_PIPE_MODE_SELECT

Source:	VideoCS
Length Bias:	2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The workload for the HCP is based upon a single frame decode. There are no states saved between frame decodes in the HCP. Once the bit stream DMA is configured with the HCP_BSD_OBJECT command, and the bitstream is presented to the HCP, the frame decode will begin.
The HCP_PIPE_MODE_SELECT command is responsible for general pipeline level configuration that would normally be set once for a single stream encode or decode and would not be modified on a frame workload basis.
This is a picture level state command and is shared by both encoding and decoding processes.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline Type</b><table><tr><td>Default Value:</td><td>2h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:23</td><td><b>Media Instruction Opcode</b><table><tr><td>Default Value:</td><td>7h Codec/Engine Name</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table>Codec/Engine Name = HCP = 7h</td></tr>
<tr><td></td><td>22:16</td><td><b>Media Instruction Command</b><table><tr><td>Default Value:</td><td>0h HCP_PIPE_MODE_SELECT</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>(Excludes Dwords 0, 1).<table><tr><th>Value</th><th>Name</th></tr><tr><td>5h</td><td>Value_5</td></tr></table></td></tr>
<tr><td>1</td><td>31:24</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>23</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>22:20</td><td><b>Reserved</b></td></tr>
</table>

HCP_PIPE_MODE_SELECT

<table>
<tr><td></td><td></td><td><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>19</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>18</td><td><b>Prefetch Disable</b>
When memory compression is enabled, we are seeing drop in performance. To compensate loss of performance due to latencies, we are adding pre-fetch.
This bit would disable prefetches if they cause unintended behavior.</td></tr>
<tr><td></td><td>17</td><td><b>Tile Based Engine</b>
This bit indicates HW works as a Tile Based Engine(as opposed to Frame based)meaning HW will flush out bitstream and streamout data to memory at the end of each Tile.
Tiling must be enabled to set this bit to 1.
If this bit set to 1, the Tile row CAN be repeated if needed.
Only current tile row is allowed to be repeated otherwise SW has to repeat all Tile Rows starting from the top tile row in a frame.</td></tr>
<tr><td></td><td>16:15</td><td><b>Pipe working Mode</b>
This programs the working mode for HCP pipe.<table><tr><th>Value</th><th>Name</th><th>Description</th><th>Programming Notes</th></tr><tr><td>00b</td><td>Legacy decoder/encoder mode (Single pipe)</td><td>This is for single pipe mode standalone mode. It is used by both decoder and encoder.</td><td></td></tr><tr><td>01b</td><td>CABAC FE only decode mode (Single CABAC pipe)</td><td>This is for the single CABAC FE only in decoder mode. This will be only run CABAC and streamout syntax element.</td><td></td></tr><tr><td>10b</td><td>Decoder BE only or Encoder mode (Scalable Multi-pipe)</td><td>This is for multiple-pipe scalable mode. In decoder, it is only on BE reconstruction. In encoder, it is for PAK.</td><td></td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">HCP_PIPE_MODE_SELECT</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>11b</td><td>Decoder Scalable mode with CABAC in real tiles (Scalable Multi-pipe)</td><td>This is for multiple-pipe scalable mode decoder mode in real tiles. CABAC and reconstruction will run together. Each pipes will run in real tiles vertically.</td><td>The real-tile/virtual tile decoding are supported for following features:
<table>
<tr><th>Feature</th><th>Virtual tile decoding</th><th>Real tile decoding</th></tr>
<tr><td>Rext HEVC (including main, main10)</td><td>yes</td><td>yes</td></tr>
<tr><td>SCC HEVC</td><td>no</td><td>yes</td></tr>
<tr><td>VP9</td><td>yes</td><td>no</td></tr>
<tr><td>AV1</td><td>no</td><td>yes</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td></td><td>14:13</td><td><b>Multi-Engine Mode</b>
This indicates the current pipe is in single pipe mode or if in scalable mode is in left/right/middle pipe in multi-engine mode.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>00b</td><td>Single Engine Mode or CABAC FE only decode mode</td><td>This is for single engine mode (legacy) OR CABAC FE only decode mode
During HEVC Decoder Scalability Real Tile Mode, for the last phase, it is possible to have single tile column left. In this case, it should be programmed with pipe as a single engine mode (using this value).
For example, for 9 tile column running on 4 pipes. The first two phases will use all 4 pipes and finish 8 tile column. The remaining one column will be processed as last third phase as single tile column.</td></tr>
<tr><td>01b</td><td>Pipe is the left engine in a Multi-engine mode</td><td>Current pipe is the most left engine while running in scalable multi-engine mode</td></tr>
<tr><td>10b</td><td>Pipe is the right engine in a Multi-engine mode</td><td>Current pipe is the most right engine while running in scalable multi-engine mode</td></tr>
<tr><td>11b</td><td>Pipe is one of the middle engine in a Multi-engine mode</td><td>Current pipe is in one of the middle engine while running in scalable multi-engine mode</td></tr>
</table>
</td></tr>
<tr><td></td><td>12</td><td><b>PAK Frame Level StreamOut enable</b>
This bit is valid if global bit PAK Pipeline Streamout Enable is set to 1. This bit is defined to use legacy tests on HW and it's valid for both hevc/vp9.
Frame level streamouts consists of 3 parts:
LCU Streamout (Set PAK Frame level streamout enable and PAK Pipeline Streamout Enable)
SSE Streamout (Set SSE enable and PAK Pipeline Streamout Enable)
RhoDomain Streamout (Set RhoDomain enable and PAK Pipeline Streamout Enable)
If set to 1, HW will output LCU streamouts which are not validated.</td></tr>
</table>

<table>
<tr><th colspan="3">HCP_PIPE_MODE_SELECT</th></tr>
<tr><td></td><td></td><td>By default it should be '0'</td></tr>
<tr><td></td><td>11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>9</td><td><b>Advanced Rate Control Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table><table><tr><th>Description</th></tr><tr><td>It is only defined for encode.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Disable</td><td>Use the legacy HW generated delta QP for multipass</td></tr><tr><td>1</td><td>Enable</td><td>Use the rate control (HW continues to generate the legacy delta QP and write to MMIO, but do not add to the final QP in the next pass)</td></tr></table></td></tr><tr><td>HW assistance- HW adds delta QP for every CU in multipass.</td></tr><tr><td>VP9: Scalability mode use only Advanced Rate Control for BRC (no HW involvement)
HEVC: Scalability mode use only HW assisted Advanced Rate Control for BRC.</td></tr></table></td></tr>
<tr><td></td><td>8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:5</td><td><b>Codec Standard Select</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>HEVC</td></tr><tr><td>1</td><td>VP9</td></tr></table></td></tr>
<tr><td></td><td>4</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>3</td><td><b>Pic Status/Error Report Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Disable</td><td>Disable status/error reporting</td></tr><tr><td>1</td><td>Enable</td><td>Status/Error reporting is written out once per picture. The Pic Status/Error Report ID in DWord3along with the status/error status bits are packed into one cache line and written to the Status/Error Buffer address in the HCP_PIPE_BUF_ADDR_STATE command. Must be zero for encoder mode.</td></tr></table></td></tr>
<tr><td></td><td>2</td><td><b>PAK Pipeline Streamout Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">HCP_PIPE_MODE_SELECT</th></tr>
<tr><td></td><td></td><td>Pipeline Streamout Enable is only defined for encode. It is ignored for decode.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Disable pipeline states and parameters streamout</td></tr>
<tr><td>1</td><td>Enable pipeline states and parameters streamout</td></tr>
</table></td></tr>
<tr><td></td><td>1</td><td><b>Deblocker Streamout Enable</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
Deblocker Streamout Enable not currently supported for Encode or Decode
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Disable</td><td>Disable deblocker-only parameter streamout</td></tr>
<tr><td>1</td><td>Enable</td><td>Enable deblocker-only parameter streamout</td></tr>
</table></td></tr>
<tr><td></td><td>0</td><td><b>Codec Select</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Decode</td></tr>
<tr><td>1</td><td>Encode</td></tr>
</table></td></tr>
<tr><td>2</td><td>31:0</td><td><b>Media Soft-Reset Counter (per 1000 clocks)</b>
<table>
<tr><td>Format:</td><td>U32</td></tr>
</table>
In decoder modes, this counter value specifies the number of clocks (per 1000) of GAC in activity before a media soft-reset is applied to the HCP. If counter value is set to 0, the media soft-reset feature is disabled and no reset will occur.
In encoder modes, this counter must be set to 0 to disable media soft reset. This feature is not supported for the encoder.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Disable</td></tr>
</table></td></tr>
<tr><td>3</td><td>31:0</td><td><b>Pic Status/Error Report ID</b>
<table>
<tr><td>Format:</td><td>U32</td></tr>
</table>
The Pic Status/Error Report ID is a unique 32-bit unsigned integer assigned to each picture status/error output. Must be zero for encoder mode.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>32-bit unsigned</td><td>Unique ID Number</td></tr>
<tr><td>1</td><td>Reserved</td><td></td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Software must program different Status/Error Buffer addresses between pictures; otherwise the hardware might overwrite previously written data.</td></tr>
</table></td></tr>
<tr><td>4</td><td>31:0</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">HCP_PIPE_MODE_SELECT</th></tr>
<tr><td>5</td><td>31:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>6</td><td>31:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7</td><td><b>Source Pixel PreFetch Enable</b>
Enables source pixel prefetch.
When set, PAK makes one request for every few LCUs (prefetch length) to warm up TLBs before actual requests are made
There is no data return so no increase in data BW.
This bit is used for HEVC in PAK only Mode
Default: Enable</td></tr>
<tr><td></td><td>6:4</td><td><b>Source Pixel Prefetch Length</b><table><tr><td>This field indicates how often (number of LCUs)PAK should make prefetch request for source pixel.
ValidRange:4-7and mapped as100->2, 101->4, 110->8 and 111->16 LCUs
This field is valid when Source Pixel PreFetch Enabled
Default Value:101 (4 LCUs)
This bit is used for HEVC in PAKonly Mode</td></tr><tr><td><u>Recommended prefetch lengths below:</u><table>
<tr><th rowspan="3">LCU/SB size</th><th colspan="4">Bits per pixel format</th></tr>
<tr><th>4:2:0 NV12
4:2:0 P010ALT</th><th>4:2:0 P010/6
4:2:2 YUY2</th><th>4:2:2 Y216
4:4:4 AYUV
4:4:4 Y410</th><th>4:4:4 Y416</th></tr>
<tr><th>8 bpp</th><th>16 bpp</th><th>32 bpp</th><th>64 bpp</th></tr>
<tr><td><b>16x16</b></td><td>Length = (64) 16 = 0x7</td><td>Length =(32) 16 = 0x7</td><td>Length =16 = 0x7</td><td>Length =8 = 0x6</td></tr>
<tr><td><b>32x32</b></td><td>Length =(32) 16 = 0x7</td><td>Length =16 = 0x7</td><td>Length =8 = 0x6</td><td>Length =4 = 0x5</td></tr>
<tr><td><b>64x64</b></td><td>Length =16 = 0x7</td><td>Length =8 = 0x6</td><td>Length =4 = 0x5</td><td>Length =2 = 0x4</td></tr>
</table></td></tr></table></td></tr>
<tr><td></td><td>3</td><td><b>Frame reconstruction disable</b>
This bit disables writing out final reconstructed pixels to memory
Normally used for B-frame as it's not used for reference purpose in encoder mode
Default value should be '0'</td></tr>
</table>

<table>
<tr><th colspan="3">HCP_PIPE_MODE_SELECT</th></tr>
<tr><td></td><td>2</td><td><b>HEVC Separate Tile Programming</b>
This indicates each tile should be programmed separately in single pipe mode. (Tile can have multiple slices. But in this case, the slice must end at end of tile so it does not affect this bit). If there are multiple tiles in a slice, the slice needs to split into each individual tiles and programmed each tiles separately).
This should be set when the following is met:
(tiles_enabled_flag =="1") && ((pps_curr_pic_ref_enabled_flag == "1") | (palette_mode_enabled_flag == "1") | (entropy_coding_sync_enabled_flag == 1))</td></tr>
<tr><td></td><td>1:0</td><td><b>Phase Indicator</b>
This is used to indicate whether this is first, middle or last phase of programming during Real-Tile Decoder Mode. Since HEVC can have up to 20 tile columns, maximum 10 phases are possible during 2 VDbox scalable mode. This is used by hardware to know if the current programming is first or last phases.
This field is ignored (programmed to 0) for other modes other than HEVC Real-Tile Decoder Mode.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>First Phase</td></tr>
<tr><td>1</td><td>Middle Phase</td></tr>
<tr><td>2</td><td>Last Phase</td></tr>
</table></td></tr>
</table>

HCP_QM_STATE

HCP_QM_STATE

Source:	VideoCS
Length Bias:	2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HCP_QM_STATE command loads the custom HEVC quantization tables into local RAM and may be issued up to20 times: 3x Colour Component plus 2x intra/inter plus 4x SizeID minus 4 for the 32x32 chroma components.
When the scaling_list_enable_flag is set to disable, the scaling matrix is still sent to the decoder, and with all entries programmed to the same value = 16.
This is a picture level state command and is issued in both encoding and decoding processes.

Dwords 2-17 form a table for the DCT coefficients, 4 8-bit coefficients/DWord.

- Size 4x4 for SizeID0, DWords 2-5.
- Size 8x8 for SizeID1/2/3, DWords 2-17.

SizeID 0 (Table 4-10)

4x4	[31:24]	[23:16]	[15:8]	[7:0]
DWord 2	AC(0,3)	AC(0,2)	AC(0,1)	DC
DWord 3	AC(1,3)	AC(1,2)	AC(1,1)	AC(1,0)
DWord 4	AC(2,3)	AC(2,2)	AC(2,1)	AC(2,0)
DWord 5	AC(3,3)	AC(3,2)	AC(3,1)	AC(3,0)

SizeID 1, 2, 3 (Table 4-11)

8x8	[31:24]	[23:16]	[15:8]	[7:0]	[31:24]	[23:16]	[15:8]	[7:0]
DWord 3,2	AC(0,7)	AC(0,6)	AC(0,5)	AC(0,4)	AC(0,3)	AC(0,2)	AC(0,1)	DC
DWord 5,4	AC(1,7)	AC(1,6)	AC(1,5)	AC(1,4)	AC(1,3)	AC(1,2)	AC(1,1)	AC(1,0)
DWord 7,6	AC(2,7)	AC(2,6)	AC(2,5)	AC(2,4)	AC(2,3)	AC(2,2)	AC(2,1)	AC(2,0)
DWord 9,8	AC(3,7)	AC(3,6)	AC(3,5)	AC(3,4)	AC(3,3)	AC(3,2)	AC(3,1)	AC(3,0)
DWord 11,10	AC(4,7)	AC(4,6)	AC(4,5)	AC(4,4)	AC(4,3)	AC(4,2)	AC(4,1)	AC(4,0)
DWord 13,12	AC(5,7)	AC(5,6)	AC(5,5)	AC(5,4)	AC(5,3)	AC(5,2)	AC(5,1)	AC(5,0)
DWord 15,14	AC(6,7)	AC(6,6)	AC(6,5)	AC(6,4)	AC(6,3)	AC(6,2)	AC(6,1)	AC(6,0)
DWord 17,16	AC(7,7)	AC(7,6)	AC(7,5)	AC(7,4)	AC(7,3)	AC(7,2)	AC(7,1)	AC(7,0)

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode

<table>
<tr><th colspan="3">HCP_QM_STATE</th></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline Type</b><table><tr><td>Default Value:</td><td>2h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:23</td><td><b>Media Instruction Opcode</b><table><tr><td>Default Value:</td><td>7h Codec/Engine Name</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table>Codec/Engine Name = HCP = 7h</td></tr>
<tr><td></td><td>22:16</td><td><b>Media Instruction Command</b><table><tr><td>Default Value:</td><td>4h HCP_QM_STATE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>Dword Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>(Excludes Dwords 0, 1).<table><tr><th>Value</th><th>Name</th></tr><tr><td>10h</td><td></td></tr></table></td></tr>
<tr><td>1</td><td>31:13</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>12:5</td><td><b>DC Coefficient</b><table><tr><td>Format:</td><td>U8</td></tr></table>Specifies the 8-bit DC coefficient for SizeID 2 and 3.<table><tr><th>Programming Notes</th></tr><tr><td>The DC Coefficient must be set to zero for SizeID 0 and 1.
The DC Coefficient must be set to scaling_list_dc_coef_minus8 + 8 for SizeID 2 and 3.</td></tr></table></td></tr>
<tr><td></td><td>4:3</td><td><b>Color Component</b><table><tr><td>Format:</td><td>U2</td></tr></table>Encoder: When RDOQ is enabled, scaling list for all 3 color components must be same. So this field is set to always 0.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Luma</td></tr><tr><td>1</td><td>Chroma Cb</td></tr><tr><td>2</td><td>Chroma Cr</td></tr><tr><td>3</td><td>Reserved</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">HCP_QM_STATE</th></tr>
<tr><td rowspan="2"></td><td>2:1</td><td><b>SizeID</b>
<table><tr><td>Format:</td><td>U2</td></tr></table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>4x4</td><td></td></tr>
<tr><td>1</td><td>8x8</td><td></td></tr>
<tr><td>2</td><td>16x16</td><td></td></tr>
<tr><td>3</td><td>32x32</td><td>(Illegal Value for Colour Component Chroma Cr and Cb.)</td></tr>
</table></td></tr>
<tr><td>0</td><td><b>Prediction Type</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Intra</td></tr>
<tr><td>1</td><td>Inter</td></tr>
</table></td></tr>
<tr><td>2..17</td><td>511:0</td><td><b>QuantizerMatrix</b></td></tr>
</table>

HCP_REF_IDX_STATE

HCP_REF_IDX_STATE

Source:	VideoCS
Length Bias:	2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This is a slice level command used in both encoding and decoding processes. For decoder, it is issued withthe HCP_BSD_OBJECT command.

Unlike AVC, HEVC allows 16 reference idx entries in each of the L0 and L1 list for a progressive picture. Hence, a max total 32 reference idx in both lists together. The same when the picture is a field picture. Regardless the number of reference idx entries, there are only max 8 reference pictures exist at any one time. Multiple reference idx can point to the same reference picture and can optionally pic a top or bottom field, or frame.

For P-Slice, this command is issued only once, representing L0 list. For B-Slice, this command can be issued up to two times, one for L0 list and one for L1 list.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 12h HCP_REF_IDX_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 10h Name:

HCP_REF_IDX_STATE		
1	31:5	**Reserved** Access: RO Format: MBZ
	4:1	**num_ref_idx_l[RefPicListNum]_active_minus1** Format: U4 num_ref_idx_l[*RefPicListNum*]_active_minus1 **Value** | **Name** [0-14] |
	0	**RefPicListNum** Format: U1 **Value** | **Name** 0 | Reference Picture List 0 1 | Reference Picture List 1
2..17	511:0	**Entries** Format: HCP_REF_LIST_ENTRY[16]

HCP_SFC_LOCK

HCP_SFC_LOCK	
Source:	BSpec
Length Bias:	2

Description
This command is used for VD/VE box to communicate with SFC before the start of any SFC workload. VD/VE uses this command to make sure that it has the ownership of SFC pipe before running workload with SFC since SFC is shared between VD/VE on a frame level. For VD(MFX)-SFC workload, only decoder mode is allowed. Encoder mode cannot use SFC.
For VD(HCP)-SFC workload, only decoder mode is allowed. Encoder mode cannot use SFC

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline</b>
<table><tr><td>Default Value:</td><td>2h Media</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:23</td><td><b>Media Command Opcode</b>
<table><tr><td>Format:</td><td>OpCode</td></tr></table>
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>9h</td><td>Media HCP+SFC Mode <b>[Default]</b></td><td>For VD(HCP)+SFC mode, only decoder mode is allowed. Encoder mode cannot use SFC</td></tr></table></td></tr>
<tr><td></td><td>22:21</td><td><b>SubOpcodeA</b>
<table><tr><td>Default Value:</td><td>0h Common</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>20:16</td><td><b>SubOpcodeB</b>
<table><tr><td>Default Value:</td><td>0h SFC Lock</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b>
<table><tr><td>Default Value:</td><td>0h Excludes DWord (0,1)</td></tr><tr><td>Format:</td><td>=n</td></tr></table>
Total Length - 2</td></tr>
</table>

HCP_SFC_LOCK		
1	31:1	**Reserved** Access: RO Format: MBZ
	0	**HCP SFC pipe select** Default Value: 1

HCP_SFC_STATE

HCP_SFC_STATE

Source: BSpec

Length Bias: 2

This command is sent from VDBOX/HCP/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Default Value: 9h Media HCP+SFC Mode Format: OpCode
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 1h SFC_State Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 2Bh Excludes DWord (0,1) Format: =n Total Length - 2
1..60	1919:0	**SFC State Body** Format: SFC_STATE_BODY

HCP_SURFACE_STATE

HCP_SURFACE_STATE

Source:	VideoCS
Length Bias:	2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HCP_SURFACE_STATE command is responsible for defining the frame buffer pitch and the offset of the chroma component.
This is a picture level state command and is shared by both encoding and decoding processes.
Note : When NV12/P010 and Tile Y are being used, full pitch and interleaved UV is always in use. U and V X offset must be set to 0; U and V Y offset must be 4-pixel aligned. For 10-bit pixel, P010 surface definition is being used.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 1h HCP_SURFACE_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value | Name 1h |
1	31:28	**Surface Id** Format: U4

HCP_SURFACE_STATE

Value	Name	Description
0h	HEVC: For current decoded Picture	8-bit uncompressed data
1h	Source Input Picture (encoder)	8-bit uncompressed data
2h	Prev Reference Picture	(VP9 only) Previous Reference
3h	Golden Reference Picture	(VP9 only) Golden Reference
4h	AltRef Reference Picture	(VP9 only) AltRef Reference
5h	HEVC: Reference Pictures	(HEVC only) Reference. Also, this will have separate compressible bits per reference surfaces for HEVC

27:17 **Reserved**

Access:	RO
Format:	MBZ

16:0 **Surface Pitch Minus1**

Format:	U17-1

This field specifies the surface pitch in (#Bytes - 1).

Programming Notes
For TileYF and TileYS surfaces, the range is dependent on the Cu parameter (refer to Memory Data Formats section for the definition of the Cu parameter depending on the case). The range in bytes is [2^{Cu}-1, 131071] -> [(2^{Cu})B, 128KB] = [1 tile, 128KB/($2^{Cu)\ tiles}$] The field specifies the surface pitch in (#Bytes - 1)
For tiled surfaces, the pitch must be a multiple of the tile width (i.e.128 bytes aligned). If Half Pitch for Chroma is set, this field must be a multiple of two tile widths for tiled surfaces, or a multiple of 2 bytes for linear surfaces. For Y-tiled surfaces: Range = [127, 131071] to [128B,128KB] = [1 tile, 1024 tiles]

2 31:27 **Surface Format**

Format:	U5

Specifies the format of the surface.

Value	Name	Description	Programming Notes
0h	YUY2 format		
1h	RGB_8 format		
2h	AYUV4444 format		
3h	P010Variant		P010Variant is a modified P010 format, >8 bit planar 420 with MSB together and LSB at an offset in x direction where the x-offset

HCP_SURFACE_STATE

			should be 32-bit aligned.
4h	PLANAR_420_8		
5h	YCRCB_SwapY format		
6h	YCRCB_SwapUV format		
7h	YCRCB_SwapUVY format		
8h	Y216/Y210 format	Same value is used to represent Y216 and Y210	
9h	RGB_10 format		
Ah	Y410 format		
Bh	NV21 Planar_420_8 Format		
Ch	Y416 format		
Dh	P010		
11h	Y216Variant	Y216Variant is the modified Y210/Y216 format, 8 bit planar 422 with MSB bytes packed together and LSB bytes at an offset in the X-direction where the x-offset is 32-bit aligned. The chroma is UV interleaved with identical MSB and LSB split as luma and is at an offset in the Y-direction (similar to NV12) but is the same height as the luma.	
12h	Y416Variant	Y416Variant is the modified Y410/Y412/Y416 format,8 bit planar 444 with MSB bytes packed together and LSB bytes at an offset in the X-direction where the x-offset is 32-bit aligned. The U channel is below the luma, has identical MSB and LSB split as luma and is at an offset in the Y-direction (similar to NV12) but is the same height as the luma The V channel is below the U, has identical MSB and LSB split as luma and is at an offset in the Y-direction (similar to NV12) but is the same height as the luma.	
13h	YUY2Variant	YUY2Variant is the modified YUY2 format, 8 bit planar 422. The chroma is UV interleaved and is at an offset in the Y-	

		direction (similar to NV12) but is the same height as the luma.	
14h	AYUV4444Variant	AYUV4444Variant is the modified AYUV4444 format, 8 bit planar 444 format. The U channel is below the luma and is at an offset in the Y-direction (similar to NV12) but is the same height as the luma. The V channel is below the and is at an offset in the Y-direction (similar to NV12) but is the same height as the luma.	
15h-1Fh	Reserved		

Programming Notes

Programming restriction on HEVC decoder:

1. If both luma_bitdepth_minus8 and chroma_bitdepth_minus 8 are both 0 (8 bits for both luma/chroma), this should be programmed to PLANAR_420_8
2. If either luma_bitdepth_minus8 or chroma_bitdepth_minus 8 is non-zero (9 or 10 bits for either or both luma/chroma), this should be programmed to P010.

26 **Reserved**

Access:	RO
Format:	MBZ

25 **Reserved**

Access:	RO
Format:	MBZ

24:15 **Reserved**

Access:	RO
Format:	MBZ

14:0 **Y Offset for U(Cb) in pixel**

Format:	U15

This field specifies the vertical offset in rows from the **Surface Base Address** to the start(origin) of the U(Cb) plane or the interleaved UV plane if **Interleave Chroma** is enabled. This field is only used for PLANAR surface formats.

Programming Notes

- For PLANAR_420 surface formats, the alignment of this field follows the tile mode described in bits 14:13 of the **Memory Address Attributes** table.
- TileY (legacy 4k) - 8 pixel aligned
- TileYF (New 4k) - 64 pixel aligned

HCP_SURFACE_STATE

<table>
<tr><td></td><td></td><td><ul><li>TileYS (64k) - 256 pixel aligned</li></ul></td></tr>
<tr><td>3</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Default Alpha Value</b></td></tr>
<tr><td>4</td><td>31:21</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>20:16</td><td><b>Compression format</b><table><tr><td>Format:</td><td><b>Media Compression Format</b></td></tr><tr><td>Format:</td><td><b>Render Compression Format</b></td></tr></table>Specifies the compression format to be used.</td></tr>
<tr><td></td><td>15:8</td><td><b>Compression Type</b>
This field indicates if the compression type for the reference surface is media or render compressed.
In HEVC mode, each bit is used for 1 reference starting with Bit 8 for Ref 0 in the ref list and Bit 9for Ref 1 and so on.
In VP9 mode, Bit 8is for Previous Reference; Bit 9is for Golden Reference and Bit 10is for Alternate Reference; Bits 11-15 are unused and should be programmed to 0<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>In VP9 mode, surface ID 2h, 3h and 4h are for previous, golden and alternate reference (3 surface states are sent per surface since the reference surfaces can be different sizes with different pitch). During all 3 surface states, this field must be programmed the same.</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>Memory Compression Enable</b>
In HEVC mode, each bit is used for 1 reference starting with Bit 0 for Ref 0 in the ref list and Bit 1 for Ref 1 and so on.
In VP9 mode, Bit 0 is for Previous Reference; Bit 1 is for Golden Reference and Bit 2 is for Alternate Reference; Bits 3-7 are unused and should be programmed to 0.<table><tr><th>Value</th><th>Name</th></tr><tr><td>1</td><td>Memory Compression Enable</td></tr><tr><td>0</td><td>Memory Compression Disable</td></tr></table></td></tr>
</table>

HCP_SURFACE_STATE

		Programming Notes In VP9 mode, surface ID 2h, 3h and 4h are for previous, golden and alternate reference (3 surface states are sent per surface since the reference surfaces can be different sizes with different pitch). During all 3 surface states, this field must be programmed the same.

HCP_TILE_CODING

HCP_TILE_CODING	
Source:	BSpec
Length Bias:	2

Programming Notes
This command is used for both HEVC and VP9 codecs

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: Opcode
	28:27	**Pipeline Type** Default Value: 2h Format: Opcode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: Opcode
	22:16	**Media Instruction Command** Default Value: 15h HCP_TILE_CODING Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n Description: Excludes Dwords 0 & 1 Value: 11h; Name:
1	31:16	**Reserved**
	15:10	**Reserved MBZ**
	9	**Tile Column store Select** This bit is used for computing Tile Column store write offset and Tile read column store read address.

HCP_TILE_CODING

Programming Notes

Tile Configuration 1: 3x3 tiles

1	2	3
4	5	6
7	8	9

In the above tiling configuration,

For Tiles 1,4,7,3,6,9Tile Columnstore select should be programmed as zero.

For Tiles 2,5,8 Tile Row store select should be programmed as 1.

Tile Configuration 1: 3x5 tiles

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

In the above tiling configuration,

For Tiles 1,6,11, 3,8, 13, 5, 10, 15Tile Columnstore select should be programmed as zero.

For Tiles 2,7, 12, 4, 9, 14Tile Columnstore select should be programmed as 1.

Tile Configuration 1: 3x5tiles

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15

In the above tiling configuration,

For Tiles 1,4,7,10,13, 3,6, 9, 12, 15 Tile Columnstore select should be programmed as zero.

For Tiles 2,5,8,11,14Tile Columnstore select should be programmed as 1.

8

Tile Row store Select

This bit is used for computingTile row store write offset and Tile read row store read address.

Programming Notes

Tile Configuration 1: 3x3 tiles

1	2	3
4	5	6
7	8	9

In the above tiling configuration,

For Tiles 1,2,3,7,8,9 Tile Row store select should be programmed as zero.

For Tiles 4,5,6 Tile Row store select should be programmed as 1.

Tile Configuration 1: 3x5 tiles

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

In the above tiling configuration,

For Tiles 1,2,3,4,5,11,12,13,14,15Tile Row store select should be programmed as zero.

<table>
<tr><th colspan="3">HCP_TILE_CODING</th></tr>
<tr><td></td><td></td><td>For Tiles 6,7,8,9,10Tile Row store select should be programmed as 1.
<b>Tile Configuration 1: 3x5tiles</b>
<table>
<tr><td>1</td><td>2</td><td>3</td></tr>
<tr><td>4</td><td>5</td><td>6</td></tr>
<tr><td>7</td><td>8</td><td>9</td></tr>
<tr><td>10</td><td>11</td><td>12</td></tr>
<tr><td>13</td><td>14</td><td>15</td></tr>
</table>
In the above tiling configuration,
For Tiles 1,2,3,7,8,9,13,14,15Tile Row store select should be programmed as zero.
For Tiles 4,5,6,10,11,12 Tile Row store select should be programmed as 1.</td></tr>
<tr><td></td><td>7:0</td><td><b>Number of Active BE Pipes</b>
Indicates the number of active, consecutive positioned Scalable VDBOXs to be used for the current frame decoding or encoding.
BE Pipe partitioning, SW must guarantee the minimum width is at least two full LCUs for each tiles
This field in general should be smaller or equal to Num of Tile columns in a Frame.
This field is ignored by HW
This field is not used by HW
<table>
<tr><th>Value</th><th>Comment</th></tr>
<tr><td>0</td><td>ignored</td></tr>
<tr><td>1</td><td>ignored</td></tr>
<tr><td>2</td><td>Supported by Encoder / Decoder.</td></tr>
<tr><td>3</td><td>Supported only by Decoder.</td></tr>
<tr><td>4</td><td>Supported only by Encoder</td></tr>
</table></td></tr>
<tr><td>2</td><td>31</td><td><b>IsLastTileOfColumn</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
Indicates if current Tile is last tile of a Column</td></tr>
<tr><td></td><td>30</td><td><b>IsLastTileOfRow</b>
Indicates if current Tile is Last Tile of a Row</td></tr>
<tr><td></td><td>29:26</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>25:16</td><td><b>Tile Row Position</b>
<table><tr><td>Format:</td><td>U10</td></tr></table>
Ctb row position of tile
For VP9: In units of SB64x64</td></tr>
</table>

HCP_TILE_CODING

	15:11	**Reserved** Access: RO Format: MBZ
	10	**Reserved** Access: RO Format: MBZ
	9:0	**Tile Column Position** Format: U10 Ctb column position of tile For VP9: In units of SB64x64
3	31	**LastPassOfTile (ValidationOnly)** This bit indicates last pass of a Tile. This is validation use only. HW/SW should not use in design
	30:27	**Reserved** Access: RO Format: MBZ
	26:16	**TileWidthInMinCbMinus1** Specifies Tile width in units of minimum coding block size. The minimal width per tile is at least two full LCUs. In HEVC Encoder mode, the following restrictions apply. Last LCU at frames right edge must align to CU boundary. This applies to all size of LCUs: 16x16, 32x32, 64x64. Kernel does not count/include CUs outside of the picture in PakObj/CU_record respectively. For VP9: In units of 8x8
	15:11	**Reserved** Access: RO Format: MBZ
	10:0	**TileHeightInMinCbMinus1** Specifies Tile Height in units of minimum coding block size In HEVC Encoder mode, the following restrictions apply. Last LCU at frames bottom edge must align to CU boundary. This applies to all size of LCUs: 16x16, 32x32, 64x64. Kernel does not count/include CUs outside of the picture in PakObj/CU_record respectively. For VP9: In units of 8x8
4	31:6	**Bitstream Byte Offset** Offset on top of base address from where the encoded bitstream should be written out for this tile In scalability mode: this offset is valid for every tile and it must be zero for the first tile in a frame. Non scalability mode: not valid
	5:1	**Reserved** Access: RO Format: MBZ

HCP_TILE_CODING

	0	**Bitstream Byte Offset Enable** 0-> Disables bitstream byte offset, meaning the encoded bitstream for all TILEs would be contiguous This bit is set to zero.
5	31:6	**PAK Frame Statistics Offset** The frame statistics (SSE, RhoDomain and LCU stats) will be reported per Tile. Valid only in scalability mode
	5:0	**Reserved** Access: RO Format: MBZ
6	31:6	**CU Level Streamout Offset** CU level statistics (see details in streamout section) pertile will be streamed out starting from this offset address This offset is valid for every Tile in scalability mode only.
	5:0	**Reserved** Access: RO Format: MBZ
7	31:6	**Slice Size Streamout Offset** Size of every slice within this tile will be streamed out starting from this offset address. This offset is valid for every Tile in scalability mode only.
	5:0	**Reserved** Access: RO Format: MBZ
8	31:6	**CU record offset** Offset address for CU record for this tile This offset is valid for every Tile in scalability mode only.
	5:0	**Reserved** Access: RO Format: MBZ
9	31:6	**SSE RowStore offset** SSE(Sum Square Error) statistics per tile will be written out at this address This offset is valid for every Tile in scalability mode.
	5:0	**Reserved** Access: RO Format: MBZ
10	31:6	**SAO RowStore offset** SAO Rowstore offset for this tile. This offset is valid for every Tile in scalability mode only.

HCP_TILE_CODING

	5:0	**Reserved** Access: RO Format: MBZ
11	31:6	**Tile Size StreamOut Offset** Tile Size will be written out at this offset This offset is valid for every Tile in scalability mode only.
	5:0	**Reserved** Access: RO Format: MBZ
12	31:6	**VP9 Probability Counter Streamout Offset** Probability counters will be written out starting from this offset address This offset is valid for every Tile in scalability mode only.
	5:0	**Reserved** Access: RO Format: MBZ
13..14	63:0	**HCP Scalability Synchronize Buffer - Base Address** Format: SplitBaseAddress64ByteAligned Specifies the 64 byte aligned buffer address used for data synchronization between neighboring pipes in scalable modes. The buffer will be written and read (as a flush mechanism) by hardware to guarantee data made it to memory before neighboring pipe can read the data. Hardware will also write the current row (in LCU) number to indicate which the current processing rows. **Programming Notes** This minimal buffer size (in CLs) should be set to the number of scalable pipes used by this workload. This data should not be cached.
15	31:0	**HCP Scalability Synchronize Buffer - Attributes** Format: MemoryAddressAttributes
16	31:0	**Reserved** Access: RO Format: MBZ
17	31:18	**Reserved** Access: RO Format: MBZ
	17:14	**Frame Number** Indicates Frame number
	13:8	**Tile number** This field indicates the tile number and used for reporting in Tile bitstream meta data.

HCP_TILE_CODING		
	7:0	**Reserved** Access: RO Format: MBZ
18	31:0	**TileMetaData_DW1** First four bytes of Meta data that goes into Tile bitstream metadata.
19	31:0	**TileMetaData_DW2** **Programming Notes** last four bytes of Meta data that goes into Tile bitstream metadata.

HCP_TILE_STATE

HCP_TILE_STATE	
Source:	VideoCS
Length Bias:	2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This command is valid for decoder only.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 11h HCP_TILE_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: Fh Name:
1	31:10	**Reserved** Access: RO Format: MBZ
	9:5	**NumTileColumnsMinus1** Format: U5 Specifies the number of tile columns in Ctbs per picture. Maximum of 20 columns are supported (level 6.2 restriction)

HCP_TILE_STATE

	4:0	**NumTileRowsMinus1** Format: U5 Specifies the number of tile rows in Ctbs per picture. Maximum of 22 rows are supported (level 6.2 restriction)
2..6	159:0	**Ctb column position of tile column** Format: **HCP_TILE_POSITION_IN_CTB[5]**
7..12	191:0	**Ctb row position of tile row** Format: **HCP_TILE_POSITION_IN_CTB[6]** Note that there are only 22 rows, so the most significant 16 bits of HCP_TILE_POSITION_IN_CTB[5] (31:16) are reserved
13..14	63:0	**Ctb column position MSB** Format: **HCP_TILE_POSITION_IN_CTB_MSB**
15..16	63:0	**Ctb row position MSB** Format: **HCP_TILE_POSITION_IN_CTB_MSB**

HCP_VP9_PIC_STATE

<table>
<tr><th colspan="3">HCP_VP9_PIC_STATE</th></tr>
<tr><td colspan="3">Source: VideoCS
Length Bias: 2</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline Type</b><table><tr><td>Default Value:</td><td>2h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:23</td><td><b>Media Instruction Opcode</b><table><tr><td>Default Value:</td><td>7h Codec/Engine Name</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table>Codec/Engine Name = HUC = Bh</td></tr>
<tr><td></td><td>22:16</td><td><b>Media Instruction Command</b><table><tr><td>Default Value:</td><td>30h HCP_VP9_PIC_STATE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>Dword Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>(Excludes Dwords 0, 1).<table><tr><th>Value</th><th>Name</th><th>Programming Notes</th></tr><tr><td>Bh</td><td>Decoder DW Length</td><td>Only Up to DW12 should be programmed for decoder</td></tr><tr><td>1Eh</td><td>Encoder DW Length</td><td>All DWs should be programmed for encoder</td></tr></table></td></tr>
<tr><td>1</td><td>31:30</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>29:16</td><td><b>Frame Height In Pixels Minus 1</b><table><tr><td>Format:</td><td>U14</td></tr></table>Specifies the height of the decoded picture in units of 8 pixels, which is the minimum coding block size. The decoded picture height in units of luma samples equals
(FrameHeightInMinBlocksMinus1+ 1) *8 1
<i>For Encoder Partial SB:</i></td></tr>
</table>

<table>
<tr><th colspan="3">HCP_VP9_PIC_STATE</th></tr>
<tr><td></td><td></td><td><i>Kernel, on last SB (at picture edges), splits the SB into as small as possible CUs to have picture boundaries aligned to CU boundary.</i>
<i>Kernel does not count/include CUs outside of the picture in PakObj/CU_record respectively.</i>
<i>Driver sets up a SB aligned (both in X/Y direction) surface.</i>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>[4096,16383]</td><td>4K_TO_16K</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Decoder supports 16K image and 8K video. Encoder only supports up to 8K.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0-6</td><td>Invalid (multiple of 8 pixels)</td></tr>
<tr><td>7-8191</td><td>8-8K Pixels (Decoder and Encoder)</td></tr>
<tr><td>8192-16383</td><td>8K-16K Pixels (Decoder only)</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td></td><td>15:14</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>13:0</td><td><b>Frame Width In Pixels Minus 1</b>
<table>
<tr><td>Format:</td><td>U14</td></tr>
</table>
Specifies the width of the decoded picture in units of minimum coding block size. The decoded picture width in units of luma samples equals (FrameWidthInMinBlocksMinus1+ 1) *8 1 This should be programmed to a multiple of 8 pixels minus 1.
<i>For Encoder Partial SB:</i>
<i>Kernel, on last SB (at picture edges), splits the SB into as small as possible CUs to have picture boundaries aligned to CU boundary.</i>
<i>Kernel does not count/include CUs outside of the picture in PakObj/CU_record respectively.</i>
<i>Driver sets up a SB aligned (both in X/Y direction) surface.</i>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>[4096,16383]</td><td>4K_TO_16K</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Decoder supports 16K image and 8K video. Encoder only supports up to 8K.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0-6</td><td>Invalid (multiple of 8 pixels)</td></tr>
<tr><td>7-8191</td><td>8-8K Pixels (Decoder and Encoder)</td></tr>
<tr><td>8192-16383</td><td>8K-16K Pixels (Decoder only)</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>31</td><td><b>Segment ID StreamIn Enable</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
</td></tr>
</table>

HCP_VP9_PIC_STATE

Indicates SegmentID from previous frame needs to be streamIn for Segment ID prediction

Value	Name
0	Disable
1	Enable

Programming Notes
Deocder Only:SegmentIDStreamInEnable = error_resilient_mode OR intra_only OR VP9_KEY_FRAME

30

Segment ID StreamOut Enable

Format:	Enable

Indicates SegmentID of current frame needs to be streamOut for next frame

Value	Name
0	Disable
1	Enable

Programming Notes
Deocder Only: SegmentIDStreamOutEnable = segmentation_enabled AND segmentation_update_map

29

Lossless Mode

Format:	Enable

This bitSet to indicate lossless coding mode.
In encoder mode, software has to set tx_mode to 4x4only and all tu_sizes in CU record as 4x4 for entire frame. Software also has to program such that final_qindex=0 and final_filter_level=0 following the Quant Scale and Filter Level Table in Segmentation State section. Hardware forces Hadamard Tx when this bit is set. When Lossless Mode is on, BRC has to be off.

Value	Name
0	Normal Mode
1	Loless Mode

28

Segmentation Temporal Update

Format:	Enable

Indicates whether segID is decoding from bitstream or predicted from previous frame.
In encoder Mode it should use either from previous frame or streamIn

Value	Name

HCP_VP9_PIC_STATE

0h	Decode segID from bitstream
1h	Get segID either from bitstream or from previous frame

Programming Notes
Decoder Only: For KEY_FRAME or INTRA_ONLY frame, this bit should be set to "0". Note: Driver should override this flag to "0" in KEY_FRAME or INTRA_ONLY frame even if this bit decoded from bitstream is different. This is for hardware optimization. This override does not affect bitstream decoding other than uncompressed header.

27

Segmentation Update Map

Format:	Enable

Indicates how hardware determines segmentation ID

Value	Name	Description
0h		Intra block: segment ID is zero Inter block: get segment ID from previous frame (streamIN)
1h		Intra block: decode segment ID from bitstream. Inter block: determines from segmentation_temporal_update setting

26

Segmentation Enabled

Format:	Enable

Indicate if segmentation is enabled or not

Value	Name
0h	All blocks are implied to belong to segment 0
1h	SegID determination depends on segmentation_update_map setting

25:23

Sharpness Level

Format:	U3

Specify the sharpness level, as one of regular deblocking strength control.

Programming Notes
Set to 0 to disable the use of sharpness control

22:17

Filter Level

Format:	U6

Specify the Filter level, as one of deblocking strength control

Programming Notes
Set to 0 to disable the use of level control

16

Frame Parallel Decoding Mode

Indicates if parallel decoding mode is enabled. This bit should come from Uncompressed header. Together with Error Resilient mode, they decide the value of AdaptProbabilityFlag.

Value	Name
0	Disable

HCP_VP9_PIC_STATE

1	Enable

15 **Error Resilient Mode**

Indicates if error resilient mode is enabled. This bit should come from Uncompressed header. When error resilient is 1, Frame Parallel Decoding Mode will be 1, and Refresh Frame Context will be 0.When error resilient is 0, Frame Parallel Decoding Mode and Refresh Frame Context read from bit stream. Together with Frame Parallel Decoding mode, they decide the value of AdaptProbabilityFlag.

Value	Name
0	Disable
1	Enable

14 **Refresh Frame Context**

Indicates if Frame Context should be refresh. This bit should come from Uncompressed header

Value	Name
0	Disable
1	Enable

Programming Notes
Decoder Only

13 **Last Frame Type**

It indicates the frame type of previous frame (Key or Non-Key Frame)

Value	Name
0	Key Frame
1	Non Key Frame

Programming Notes
Not used in Encoder Mode

12 **Selectable TX Mode**

Format:	U1

Indicates if tx_mode is selectable

Value	Name
0	Encoder does not pack tu_size into bitstream. This helps reduce bitstream size further.
1	Encoder packs tu_size into bitstream.

Programming Notes
HW always picks tu_size from CU record of pak_obj. SW responsibility to set tu_size correct.

HCP_VP9_PIC_STATE

11

Hybrid Prediction Mode

Format:	U1

Indicates if comp_pred_mode is hybrid

Value	Name
0	comp_prediction_mode!=HYBRID, Encoder does not pack comp_pred_mode [interpred_comp in pak_obj] into bitstream.
1	comp_prediction_mode==HYBRID, Encoder packs comp_pred_mode into bitstream. This helps reduce bitstream size further.

10

Use Prev in Find MV References

Format:	Enable

0: Temporal MV buffer is not available for MV prediction 1: Temporal MV buffer is available for MV prediction This is set to 0 when: The last picture has a different size Current picture is error-resilient mode Current picture is intra_only, or keyframe Last picture was intra_only or keyframe Last picture was not a displayed picture.

9:7

Ref Frame Sign Bias[0..2]

Format:	U3

Reference Frame sign bias (not including intra reference) Bit[7] Sign Bias of Last Frame Bit[8] Sign Bias of Golden Frame Bit[9] Sign Bias of AltRef Frame

6:4

Mcomp Filter Type

Format:	U3

Indicate Motion Compensation Filter type.
If set to 4, encoder uses modes in pak_obj command.

Value	Name
0h	Eight-tap
1h	Eight-tap-Smooth
2h	Eight-tap-Sharp
3h	Bilinear
4h	Switchable

3

Allow Hi Precision MV

Format:	Enable

Indicate high precision mode for Motion Vector prediction

Value	Name
0h	Normal mode
1h	High Precision mode

2

IntraOnly Flag

Format:	Enable

Indicates intra-only for inter pics. MBZ for keyframes.

HCP_VP9_PIC_STATE

Programming Notes
Used for Non-displayable picture; SW responsibility to make sure no Inter block in pak_obj of this frame

Value	Description
0	Inter Frame use both intra/inter-blocks
1	Inter frame use only inta-blocks

1

Adapt Probabilities Flag

Format:	Enable

Indicates that the probabilities used to decode this frame should be adapted

Value	Name
0h	0: Do not adapt (error resilient or frame_parallel_mode are set)
1h	1: Adapt (not error resilient and not frame_ parallel_mode)

Programming Notes
Only forward adaptation is supported in encoder mode.

0

Frame Type

Format:	U1

Specifies the VP9 frame type

Value	Name
0h	Key Frame
1h	Inter Frame

3

31:28

Profile Level

Format:	U4

This indicates VP9 Profile level from bitstream

Value	Name	Description
0	Profile_0	Profile 0 only supports 8 bit 420 only
2	Profile_2	Profile 2 only supports 10 bits 420 only
1	Profile_1	Profile 1 only supports 8 bit 444 only
3	Profile_3	Profile 3 only supports 10-bit 444 only

Programming Notes
Profile 0, 1, 2, and 3 are supported. Profile 0: 8 bit 420 only Profile 1: 8 bit 444 (422 is NOT supported) Profile 2: 10/12 bit 420 Profile 3: 10/12 bit 444 (422 is NOT supported)

27:24

BitDepthMinus8

Format:	U4

HCP_VP9_PIC_STATE

This indicates the bitdepth (minus 8) of the pixels

Value	Name	Programming Notes
0	Bitdepth_8	It indicates pixel bitdepth is 8. Only profile 0 is allowed in this mode. It indicates pixel bitdepth is 8. Only profile 0 and 1 are allowed in this mode.
2	Bitdepth_10	It indicates pixel bitdepth is 10. Only profile 2 is allowed in this mode. It indicates pixel bitdepth is 10. Only profile 2 and 3 are allowed in this mode.
4	Bitdepth_12	It indicates pixel bitdepth is 12. Only profile 2 is allowed in this mode.

Programming Notes
In profile 0 and 1, only value of 0 (8 bit pixel) is allowed. In profile 2 and 3, only value of 2 and 4 (10 or 12 bit pixel) are allowed.

23:22 **Chroma Sampling Format**

Format:	U2

This indicates the chroma sampling format of the bitstream

Value	Name	Programming Notes
0	Format_420	Chroma Format 420, supported by profile 0 and 2
2	Format_444	Chroma Format 444, supported by Profile 1 and 3

Programming Notes
Currently only 420 and 444 are supported (in profile 0, 1, 2 and 3). All other modes are not valid. Value 0: Format 420: Profile 0 and 2 only (supported) Value 1: Format 422: Profile 1 and 3 only: Currently NOT supported Value 2: Format 444: Profile 1 and 3 only:(supported)

21 **Reserved**

20:10 **Reserved**

Access:	RO
Format:	MBZ

9:8 **Log2 Tile Row**

Format:	U2

This indicates the number of tile rows (log2).

Value	Name
0h	1 Tile Row
1h	2 Tile Row
2h	4 Tile Row

<table>
<tr><th colspan="3">HCP_VP9_PIC_STATE</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Decoder Only as encoder must use TILE_CODING_COMMAND</td></tr>
</table>
</td></tr>
<tr><td></td><td>7:4</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>3:0</td><td><b>Log2 Tile Column</b>
<table>
<tr><td>Format:</td><td>U4</td></tr>
</table>
This indicates the number of tile rows (log2).
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0h</td><td>1 Tile Column</td></tr>
<tr><td>1h</td><td>2 Tile Column</td></tr>
<tr><td>2h</td><td>4 Tile Column</td></tr>
<tr><td>3h</td><td>8 Tile Column</td></tr>
<tr><td>4h</td><td>16 Tile Column</td></tr>
<tr><td>5h</td><td>32 Tile Column</td></tr>
<tr><td>6h</td><td>64 Tile Column</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Decoder only as encoder must use TILE_CODING_COMMAND</td></tr>
</table>
</td></tr>
<tr><td>4</td><td>31:16</td><td><b>Horizontal Scale Factor for LAST</b>
<table>
<tr><td>Format:</td><td>U2.14</td></tr>
</table>
This indicates the scaling factor between current frame and the last reference frame Set to (LastWidth * 2^14)/CurrentWidth
</td></tr>
<tr><td></td><td>15:0</td><td><b>Vertical Scale Factor for LAST</b>
<table>
<tr><td>Format:</td><td>U2.14</td></tr>
</table>
This indicates the scaling factor between current frame and the last reference frame Set to (LastHeight * 2^14)/CurrentHeight
</td></tr>
<tr><td>5</td><td>31:16</td><td><b>Horizontal Scale Factor for GOLDEN</b>
<table>
<tr><td>Format:</td><td>U2.14</td></tr>
</table>
This indicates the scaling factor between current frame and the golden reference frame Set to (LastWidth * 2^14)/CurrentWidth
</td></tr>
<tr><td></td><td>15:0</td><td><b>Vertical Scale Factor for GOLDEN</b>
<table>
<tr><td>Format:</td><td>U2.14</td></tr>
</table>
This indicates the scaling factor between current frame and the golden reference frame Set to (LastHeight * 2^14)/CurrentHeight"
</td></tr>
<tr><td>6</td><td>31:16</td><td><b>Horizontal Scale Factor for ALTREF</b>
<table>
<tr><td>Format:</td><td>U2.14</td></tr>
</table>
This indicates the scaling factor between current frame and the altref reference
</td></tr>
</table>

HCP_VP9_PIC_STATE

		frame Set to (LastWidth * 2^14)/CurrentWidth
	15:0	**Vertical Scale Factor for ALTREF** Format: U2.14 This indicates the scaling factor between current frame and the altref reference frame Set to (LastHeight * 2^14)/CurrentHeight
7	31	**Reserved**
	30	**Reserved** Access: RO Format: MBZ
	29:16	**Last Frame Height In Pixels Minus 1** Format: U14 Specifies the height of the Last picture in units of pixels. The Last picture height in units of luma samples equals (LastFrameHeightInMinBlocksMinus1+ 1) **Value** | **Description** 0-16383 | 1-16K Pixels (encoder only goes to 8K)
	15:14	**Reserved** Access: RO Format: MBZ
	13:0	**Last Frame Width In Pixels Minus 1** Format: U14 Specifies the width of the Last picture in units of pixels. The Last picture width in units of luma samples equals (LastFrameWidthtInMinBlocksMinus1+ 1) **Value** | **Description** 0-16383 | 1-16K Pixels (encoder only goes to 8K)
8	31:30	**Reserved** Access: RO Format: MBZ
	29:16	**Golden Frame Hieght In Pixels Minus 1** Format: U14 Specifies the height of the Last picture in units of pixels. The Golden picture height in units of luma samples equals (LastFrameHeightInMinBlocksMinus1+ 1) **Value** | **Description** 0-16383 | 1-16K Pixels (encoder only goes to 8K)
	15:14	**Reserved** Access: RO Format: MBZ
	13:0	**Golden Frame Width In Pixels Minus 1**

HCP_VP9_PIC_STATE

<table>
<tr><td></td><td></td><td><table><tr><td>Format:</td><td>U14</td></tr></table>Specifies the width of the Last picture in units of pixels. The Golden picture width in units of luma samples equals (LastFrameWidthtInMinBlocksMinus1+ 1)<table><tr><th>Value</th><th>Description</th></tr><tr><td>0-16383</td><td>1-16K Pixels (encoder only goes to 8K)</td></tr></table></td></tr>
<tr><td>9</td><td>31:30</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>29:16</td><td><b>Altref Frame Height In Pixels Minus 1</b><table><tr><td>Format:</td><td>U14</td></tr></table>Specifies the height of the Last picture in units of pixels. The Altref picture height in units of luma samples equals (LastFrameHeightInMinBlocksMinus1+ 1)<table><tr><th>Value</th><th>Description</th></tr><tr><td>0-16383</td><td>1-16K Pixels (encoder only goes to 8K)</td></tr></table></td></tr>
<tr><td></td><td>15:14</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>13:0</td><td><b>Altref Frame Width In Pixels Minus 1</b><table><tr><td>Format:</td><td>U14</td></tr></table>Specifies the width of the Last picture in units of pixels. The Altref picture width in units of luma samples equals (LastFrameWidthtInMinBlocksMinus1+ 1)<table><tr><th>Value</th><th>Description</th></tr><tr><td>0-16383</td><td>1-16K Pixels (encoder only goes to 8K)</td></tr></table></td></tr>
<tr><td>10</td><td>31:16</td><td><b>First Partition Size in Bytes [15:0]</b><table><tr><td>Format:</td><td>U16</td></tr></table>Specifies the number of bytes taken up by the first partition size which handle the probability updates<table><tr><th>Programming Notes</th></tr><tr><td>Only used by Decoder</td></tr></table></td></tr>
<tr><td></td><td>15:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">HCP_VP9_PIC_STATE</th></tr>
<tr><td></td><td>7:0</td><td><b>Uncompressed Header Length in Bytes [7:0]</b>
<table><tr><td>Format:</td><td>U8</td></tr></table>
Specifies the number of bytes taken up by the uncompressed frame header.
<table><tr><th>Programming Notes</th></tr><tr><td>this field is used by decoder only</td></tr></table></td></tr>
<tr><td>11</td><td>31:4</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>3</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>2</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>1</td><td><b>Motion Comp Scaling Enable Bit</b>
This bit must be set to "1"
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>1</td><td>Enable <b>[Default]</b></td><td>This enables Motion Comp Scaling</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>12</td><td>31:0</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>13</td><td>31:26</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>25</td><td><b>Header Insertion Enable</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td></td><td>No header insertion into the output bitstream buffer, before the current slice encoded bits.</td></tr>
<tr><td>1</td><td></td><td>Header insertion into the output bitstream buffer is present, and is before the current slice encoded bits.</td></tr></table>
Must be followed by the PAK Insertion Object Command to perform the actual insertion.
For VP9: Header is always present and this bit can never be Zero.
As HW does the header back-annotation at the end of frame we currently cannot disable it, if header was not written by HW.
Media SDK sends 2 headers. One header has original header and second header has 0s for BRC parameters (LF refDelta, ModeDelta and BaseQindex). Driver needs to pick first header for the first pass, and second header for subsequent passes.</td></tr>
</table>

<table>
<tr><th colspan="3">HCP_VP9_PIC_STATE</th></tr>
<tr><td></td><td></td><td><table><tr><th>Programming Notes</th></tr><tr><td>Encoder Only</td></tr></table></td></tr>
<tr><td></td><td>24</td><td><b>Tail Insertion Enable</b><table><tr><td>Format:</td><td>U1</td></tr></table><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td></td><td>No tail insertion into the output bitstream buffer, after the current slice encoded bits.</td></tr><tr><td>1</td><td></td><td>Tail insertion into the output bitstream buffer is present, and is after the current slice encoded bits.</td></tr></table>Must be followed by the PAK Insertion Object Command to perform the actual insertion.
Tail has to be inserted only with the last slice of frame and for VP9 only at the end of Frame.<table><tr><th>Programming Notes</th></tr><tr><td>Encoder Only</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>Base Q Index (Same as Luma AC)</b><table><tr><td>Format:</td><td>U8</td></tr></table>Added to Delta Q index of Segment
Valid Values : 0..255<table><tr><th>Programming Notes</th></tr><tr><td>Encoder Only</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Compressed header BIN count</b><table><tr><td>Format:</td><td>U16</td></tr></table>Number of bins compressed header
This field is fixed insideHW<table><tr><th>Programming Notes</th></tr><tr><td>Encoder Only</td></tr></table></td></tr>
<tr><td>14</td><td>31:21</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>20:16</td><td><b>Luma DC Q Index Delta</b><table><tr><td>Format:</td><td>S4</td></tr></table>QP delta value for Luma DC
Valid Values : -15..15<table><tr><th>Programming Notes</th></tr><tr><td>Encoder Only</td></tr></table></td></tr>
<tr><td></td><td>15:13</td><td><b>Reserved</b></td></tr>
</table>

HCP_VP9_PIC_STATE

		Access: RO Format: MBZ
	12:8	**ChromaDC_QindexDelta** Format: S4 QP Delta Value For Chroma DC Valid Values : -15..15 **Programming Notes** Encoder Only
	7:5	**Reserved** Access: RO Format: MBZ
	4:0	**ChromaAC_QindexDelta** Format: S4 QP Delta Value For Chroma AC Valid Values : -15..15 **Programming Notes** Encoder Only
15	31	**Reserved** Access: RO Format: MBZ
	30:24	**LF_ref_delta3** Format: S6 Loop filter level delta3 value; valid range -6363 With this range -63..63, in the case of Base LF Level > 31 (first pass) or (Base LF Level + BRC accumulated LF Delta) > 31 (subsequent pass), the lf_ref_delta0,1,2,3 and lf_mode_delta0,1 in compressed bitstream would be in the range -31..31. **Programming Notes** Encoder Only
	23	**Reserved** Access: RO Format: MBZ
	22:16	**LF_ref_delta2** Format: S6 Loop filter level delta2 value; valid range -6363 With this range -63..63, in the case of Base LF Level > 31 (first pass) or (Base LF Level + BRC accumulated LF Delta) > 31 (subsequent pass), the lf_ref_delta0,1,2,3 and lf_mode_delta0,1 in compressed bitstream would be in the range -31..31.

HCP_VP9_PIC_STATE

		Programming Notes Encoder Only
	15	**Reserved** Access: RO Format: MBZ
	14:8	**LF_ref_delta1** Format: S6 Loop filter level delta1 value; valid range -6363 With this range -63..63, in the case of Base LF Level > 31 (first pass) or (Base LF Level + BRC accumulated LF Delta) > 31 (subsequent pass), the lf_ref_delta0,1,2,3 and lf_mode_delta0,1 in compressed bitstream would be in the range -31..31. **Programming Notes** Encoder Only
	7	**Reserved** Access: RO Format: MBZ
	6:0	**LF_ref_delta0** Format: S6 Loop filter level delta0 value; valid range -6363 With this range -63..63, in the case of Base LF Level > 31 (first pass) or (Base LF Level + BRC accumulated LF Delta) > 31 (subsequent pass), the lf_ref_delta0,1,2,3 and lf_mode_delta0,1 in compressed bitstream would be in the range -31..31. **Programming Notes** Encoder Only
16	31:15	**Reserved** Access: RO Format: MBZ
	14:8	**LF Mode Delta 1** Format: S6 Loop filter level mode1 value; valid range -6363 With this range -63..63, in the case of Base LF Level > 31 (first pass) or (Base LF Level + BRC accumulated LF Delta) > 31 (subsequent pass), the lf_ref_delta0,1,2,3 and lf_mode_delta0,1 in compressed bitstream would be in the range -31..31. **Programming Notes** Encoder Only
	7	**Reserved** Access: RO Format: MBZ

HCP_VP9_PIC_STATE

	6:0	**LF Mode Delta 0** Format: S6 Loop filter level mode1 value; valid range -6363 With this range -63..63, in the case of Base LF Level > 31 (first pass) or (Base LF Level + BRC accumulated LF Delta) > 31 (subsequent pass), the lf_ref_delta0,1,2,3 and lf_mode_delta0,1 in compressed bitstream would be in the range -31..31. **Programming Notes** Encoder Only
17	31:16	**BitOffsetForLFModeDelta** Format: U16 Offset from starting position of output bitstream in bits where LFModeDelta should be inserted. In BRC mode, always insert LFModeDelta. {This implies uncompressed header should have: mode_ref_delta_enabled=1 and mode_ref_delta_update=1} and The uncompressed header starting from this offset **BitOffsetForLFModeDelta** has to have the following 16 bits format: {Start here: 1b1, mode_delta_0[6:0], 1b1, mode_delta_1[6:0]} and **BitOffsetForLFModeDelta = BitOffsetForLFRefDelta + 32.** **Encoder Only**
	15:0	**BitOffsetForLFRefDelta** Format: U16 Offset from starting position of output bitstream in bits where LFRefDelta should be inserted. In BRC mode, always insert LFRefDelta. {This implies uncompressed header should have: mode_ref_delta_enabled=1 and mode_ref_delta_update=1} and The uncompressed header starting from this offset **BitOffsetForLFRefDelta** has to have the following 32 bits format: {Start here: 1b1, ref_delta_0[6:0], 1b1, ref_delta_1[6:0], 1b1, ref_delta_2[6:0], 1b1, ref_delta_3[6:0]}. Encoder Only
18	31:16	**BitOffsetForLFLevel** Format: U16 Offset from starting position of output bitstream in bits where LFLevel should be inserted. Encoder Only
	15:0	**BitOffsetForQindex** Format: U16 Offset from starting position of output bitstream in bits where Qindex should be inserted. Encoder Only
19	31:27	**Reserved**

HCP_VP9_PIC_STATE

Access:	RO
Format:	MBZ

26 **FrameSzUnderStatusEn - FrameBitRateMinReportMask**

Format:	U1

This is a mask bit controlling if the condition of frame level bit count is less than FrameBitRateMin.

Value	Name	Description
0	Disable	Do not update bit 2 (Frame Bit Count Violate -- under run) of HCP_VP9_IMAGE_STATUS control register.
1	Enable	Set bit 2 (Frame Bit Count Violate -- under run) of HCP_VP9_IMAGE_STATUS control register if the total frame level bit counter is less than or equal to Frame Bit Rate Minimum limit.

Programming Notes
Encoder Only

25 **FrameSzOverStatusEn - FrameBitRateMaxReportMask**

Format:	U1

This is a mask bit controlling if the condition of frame level bit count exceeds FrameBitRateMax.

Value	Name	Description
0	Disable	Do not update bit 1 of HCP_VP9_IMAGE_STATUS control register.
1	Enable	Set bit 1 of HCP_VP9_IMAGE_STATUS control register if the total frame level bit counter is greater than or equal to Frame Bit Rate Maximum limit.

Programming Notes
Encoder Only

24:18 **Reserved**

Access:	RO
Format:	MBZ

17 **Reserved**

Access:	RO
Format:	MBZ

16 **NonFirstPassFlag**

This signals the current pass is not the first pass. It will imply designate HW behavior.

<table>
<tr><th colspan="3">HCP_VP9_PIC_STATE</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Disable</td><td>If it is initial-Pass, this bit is set to 0.</td></tr>
<tr><td>1</td><td>Enable</td><td>For subsequent passes, this bit is set to 1.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Encoder Only</td></tr>
</table>
</td></tr>
<tr><td></td><td>15:0</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>20</td><td>31</td><td>
<b>FrameBitrateMaxUnit</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
This field is the Frame Bitrate Maximum Limit Units.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Byte</td><td>32byte unit</td></tr>
<tr><td>1</td><td>KiloByte</td><td>4Kbyte unit</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Encoder Only</td></tr>
</table>
</td></tr>
<tr><td></td><td>30:14</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>13:0</td><td>
<b>FrameBitRateMax</b>
<table>
<tr><td>Format:</td><td>U14</td></tr>
</table>
<table>
<tr><td>This field is the Frame Bitrate Maximum Limit. This field along with FrameBitrateMaxUnit determines maximum allowed bits in a frame before multi-pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count exceeds this value.</td></tr>
<tr><td>0-512KB The programmable range is 0-512KB when FrameBitrateMaxUnit is 0.</td></tr>
<tr><td>0-64MB The programmable range is 0-64Mbyte when FrameBitrateMaxUnit is 1.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Encoder Only</td></tr>
</table>
</td></tr>
<tr><td>21</td><td>31</td><td>
<b>FrameBitrateMinUnit</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
This field is the Frame Bitrate Maximum Limit Units.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Byte</td><td>32byte unit</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th colspan="3">HCP_VP9_PIC_STATE</th></tr>
<tr><td></td><td></td><td><table><tr><td>1</td><td>KiloByte</td><td>4Kbyte unit</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>Encoder Only</td></tr></table></td></tr>
<tr><td></td><td>30:14</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>13:0</td><td><b>FrameBitRateMin</b><table><tr><td>Format:</td><td>U14</td></tr></table><table><tr><td>This field is the Frame Bitrate Minimum Limit. This field along with FrameBitrateMinUnit determines maximum allowed bits in a frame before multi-pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count exceeds this value.</td></tr><tr><td>0-512KB The programmable range is 0-512KB when FrameBitrateMinUnit is 0.</td></tr><tr><td>0-64MB The programmable range is 0-64Mbyte when FrameBitrateMinUnit is 1.</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>Encoder Only</td></tr></table></td></tr>
<tr><td>22..23</td><td>63:0</td><td><b>FrameDeltaQindexMax</b><table><tr><td>Format:</td><td>ExtendedMessageDescriptor-SamplingEngineNon-Bindless_</td></tr></table>Frame level delta Qindex which should be used in case FrameSize - FrameBitRateMax in the range of ((FrameDeltaQindexLFMaxRange[n] * FrameBitRateMax»5)), FrameDeltaQindexLFMaxRange[n+1] * FrameBitRateMax»5)). Each DelatQindexMax value is 8-bit with S7M format<table><tr><th>Programming Notes</th></tr><tr><td>If n == 7, DeltaQpMaxRange is infinity.</td></tr><tr><td>Encoder Only</td></tr></table></td></tr>
<tr><td>24</td><td>31:0</td><td><b>FrameDeltaQindexMin</b><table><tr><td>Format:</td><td>ExtendedMessageDescriptor-SamplingEngineNon-Bindless</td></tr></table>Frame level delta Qindex which should be used in case FrameSize - FrameBitRateMin in the range of ((FrameDeltaQindexLFMinRange[n] * FrameBitRateMin»5)), FrameDeltaQindexLFMinRange[n+1] * FrameBitRateMin»5)). Each DelatQindexMin value is 8-bit with S7M format<table><tr><th>Programming Notes</th></tr><tr><td>If n == 3, FrameDeltaQindexLFMaxRange is zero. (n>3 is not supported)</td></tr><tr><td>Encoder Only</td></tr></table></td></tr>
<tr><td>25..26</td><td>63:0</td><td><b>FrameDeltaLFMax</b><table><tr><td>Format:</td><td>ExtendedMessageDescriptor-SamplingEngineNon-Bindless</td></tr></table></td></tr>
</table>

HCP_VP9_PIC_STATE

<table>
<tr><th colspan="3">HCP_VP9_PIC_STATE</th></tr>
<tr><td></td><td></td><td>Frame level delta Loop Filter Level which should be used in case FrameSize - FrameBitRateMax in the range of ((FrameDeltaQindexLFMaxRange[n] * FrameBitRateMax»5)), FrameDeltaQindexLFMaxRange[n+1] * FrameBitRateMax»5)).
Each delta_lf_max is 7 bits with S6M format
<b>[bits 7, 15, 23, 31,.63 are reserved]</b>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>If n == 7, FrameDeltaQindexLFMaxRange is infinity.</td></tr>
<tr><td>Encoder Only</td></tr>
</table></td></tr>
<tr><td>27</td><td>31:0</td><td><b>FrameDeltaLFMin</b>
<table>
<tr><td>Format:</td><td>ExtendedMessageDescriptor-SamplingEngineNon-Bindless</td></tr>
</table>
Frame level delta Loop Filter Level which should be used in case FrameSize - FrameBitRateMin in the range of ((FrameDeltaQindexLFMinRange[n] * FrameBitRateMin»5)), FrameDeltaQindexLFMinRange[n+1] * FrameBitRateMin»5)).
Each delta_lf_min is 7 bits with S6M format
<b>[bits 7, 15, 23, 31 are reserved]</b>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>If n == 3, FrameDeltaQindexLFMaxRange is zero. (n>3 is not supported)</td></tr>
<tr><td>Encoder Only</td></tr>
</table></td></tr>
<tr><td>28..29</td><td>63:0</td><td><b>FrameDeltaQindexLFMaxRange</b>
Condition: FrameDeltaQindexLFMaxRange[n] >= FrameDeltaQindexLFMaxRange[n-1] This field is to calculate ranges for Frame level delta Qindex, specifically Frame level delta Qindex[n] and Frame level delta Qindex[n+1].
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>If n == 0, FrameDeltaQindexLFMaxRange is zero.</td></tr>
<tr><td>Encoder Only</td></tr>
</table></td></tr>
<tr><td>30</td><td>31:0</td><td><b>FrameDeltaQindexLFMinRange</b>
Condition: FrameDeltaQindexLFMinRange[n] >= FrameDeltaQindexLFMinRange[n-1]This field is to calculate ranges for Frame level delta Qindex, specifically Frame level delta Qindex[n] and Frame level delta Qindex[n+1].
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>If n == 0, FrameDeltaQindexLFMinRange is zero.</td></tr>
<tr><td>Encoder Only</td></tr>
</table></td></tr>
<tr><td>31</td><td>31:30</td><td><b>MinFrameSizeUnits</b>
<table>
<tr><td>Format:</td><td>U2</td></tr>
</table>
This field is the Minimum Frame Size Units
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>4Kb</td><td>Minimum Frame Size is in 4Kbytes.</td></tr>
<tr><td>1</td><td>16Kb</td><td>Minimum Frame Size is in 4Kbytes.</td></tr>
<tr><td>2</td><td>Compatibility Mode</td><td></td></tr>
<tr><td>3</td><td>6 Bytes</td><td></td></tr>
</table></td></tr>
</table>

HCP_VP9_PIC_STATE

		Programming Notes Encoder Only
	29:16	**Reserved** Access: RO Format: MBZ
	15:0	**MinFramSize** Format: U16 Minimum Frame Size [15:0] (in Word, 16-bit)(Encoder Only) Minimum Frame Size is specified to compensate for intel Rate Control Currently zero fill (no need to perform emulation byte insertion) is done at the last slice of a picture. It is needed for CBR. Intel encoder parameter, not part of DXVA. The caller should always make sure that the value, represented by Minimum Frame Size, is always less than maximum frame size FrameBitRateMax. This field is reserved in Decode mode. **Programming Notes** Programmable range is 0..(2^16-1) * 2^12 when MinFrameSizeUnits is 0. (4KB unit) Programmable range is 0..(2^16-1) * 2^14 when MinFrameSizeUnits is 1. (16KB unit) Encoder Only
32	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**BitOffsetForFirstPartitionSize** Format: U16 Offset from starting position of output bitstream in bits where First Partition Size should be inserted. **Programming Notes** Encoder Only
33	31:16	**Class0_SSE_Threshold1** Format: U16 **Programming Notes** This field specifies the upper bound threshold for Class0 Zone1 to classify the per-4x4sblk SSE statistics. Class0_SSE_Threshold_0 < per-4x4sblk SSE <= Class0_SSE_Threshold_1 fall under Class0 Zone1. Class0_SSE_Threshold_1 < per-4x4sblk SSE fall under Class0 Zone2. Encoder Only

HCP_VP9_PIC_STATE		
	15:0	**Class0_SSE_Threshold0** Format: | U16 **Programming Notes** This field specifies the upper bound threshold for Class0 Zone0 to classify the per-4x4sblk SSE statistics. per-4x4sblk SSE <= Class0_SSE_Threshold_0 fall under Class0 Zone0. Encoder Only
34..41 **Programming Notes:** SSE thresholds for Class 1-8, see DW 33 (SSE Class 0 thresholds) for format. Encoder Only	255:0	**SSE thresholds for Class1-8** Format: | U256

HCP_VP9_SEGMENT_STATE

HCP_VP9_SEGMENT_STATE

Source: VideoCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HUC = Bh
	22:16	**Media Instruction Command** Default Value: 32h HCP_VP9_SEGMENT_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 6h Name:
1	31:3	**Reserved** Access: RO Format: MBZ
	2:0	**Segment ID** Format: U3 The segment ID of the DWORDS following this one
2	31:4	**Reserved** Access: RO Format: MBZ

HCP_VP9_SEGMENT_STATE

	3	**Segment Reference Enabled** Format: Enable For encoder: When Segment Reference Enabled is set to 1, Software (Kernel) needs to program all PUs of this segment accordingly. This means: CU record PU refframe0=Segment Reference bit[2:1], refframe1=0, and interpred_comp=single.
	2:1	**Segment Reference** Format: U2 Indicates reference frame for this segment. **Programming Notes** If the current frame is a KEY frame or INTRA_ONLY frame, this field should be set to INTRA for all segments.
	0	**Segment Skipped** Format: Enable Indicates skip mode for this segment. **Programming Notes** If set to 1, all delta coefficients and MVs within CU of this segment should be forced to zero in HW. No block less than 8x8 size allowed segmentSkipped If set to 0, skipcoeff_flag should be coded as normal
3	31:30	**Reserved** Access: RO Format: MBZ
	29:24	**FilterLevelRef1Mode1** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref1 (Last Frame) and Mode 1.
	23:22	**Reserved** Access: RO Format: MBZ
	21:16	**FilterLevelRef1Mode0** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref1 (Last Frame) and Mode 0.
	15:14	**Reserved** Access: RO Format: MBZ

HCP_VP9_SEGMENT_STATE

	13:8	**FilterLevelRef0Mode1** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref0 (Last Frame) and Mode 1.
	7:6	**Reserved** Access: RO Format: MBZ
	5:0	**FilterLevelRef0Mode0** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref0 (Last Frame) and Mode 0.
4	31:30	**Reserved** Access: RO Format: MBZ
	29:24	**FilterLevelRef3Mode1** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref3 (Last Frame) and Mode 1.
	23:22	**Reserved** Access: RO Format: MBZ
	21:16	**FilterLevelRef3Mode0** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref3 (Last Frame) and Mode 0.
	15:14	**Reserved** Access: RO Format: MBZ
	13:8	**FilterLevelRef2Mode1** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref2 (Last Frame) and Mode 1.
	7:6	**Reserved** Access: RO Format: MBZ

HCP_VP9_SEGMENT_STATE		
	5:0	**FilterLevelRef2Mode0** Exists If: //Decoder mode only Format: U6 Indicates final filter level for Ref2 (Last Frame) and Mode 0.
5	31	**Reserved** Access: RO Format: MBZ
	30:16	**Luma AC Quant Scale (Decode mode Only)** Format: U15 Indicates final value of Luma AC Quantized Scale value. Value: [4,29247] — Name: Valid_Range
	15	**Reserved** Access: RO Format: MBZ
	14:0	**Luma DC Quant Scale (Decode mode Only)** Format: U15 Indicates final value of Luma DC Quantized Scale value. Value: [4,21387] — Name: Valid_Range
6	31	**Reserved** Access: RO Format: MBZ
	30:16	**Chroma AC Quant Scale (Decode mode Only)** Format: U15 Indicates final value of Chroma AC Quantized Scale value. Value: [4,29247] — Name: Valid_Range
	15	**Reserved** Access: RO Format: MBZ
	14:0	**Chroma DC Quant Scale (Decode mode Only)** Format: U15 Indicates final value of Chroma DC Quantized Scale value. Value: [4,21387] — Name: Valid_Range

HCP_VP9_SEGMENT_STATE

7	31:23	**Reserved** Access: RO Format: MBZ
	22:16	**Segment LF Level Delta (Encode mode Only)** Format: S6 Indicates the Loop Filter Delta for this segment. Must be 0 when segmentation_enabled == 0. Range -63..63
	15:9	**Reserved** Access: RO Format: MBZ
	8:0	**Segment QIndex Delta (encode mode only)** Format: S8 Indicates the QIndex delta for this segment. Must be 0 when segmentation_enabled == 0. Range -255..255

HCP_WEIGHTOFFSET_STATE

HCP_WEIGHTOFFSET_STATE

Source:	VideoCS
Length Bias:	2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This slice level command is issued in both the encoding and decoding processes, if the weighted_pred_flag or weighted_bipred_flag equals one. If zero, then this command is not issued.
Weight Prediction Values are provided in this command. Only Explicit Weight Prediction is supported in encoder.
For P-Slice, this command is issued only once together with HCP_REF_IDX_STATE Command for L0 list. For B-Slice, this command can be issued up to two times together with HCP_REF_IDX_STATE Command, one forL0 list and one for L1 list.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 13h HCP_WEIGHTOFFSET_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 28h — Name: 40
1	31:1	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">HCP_WEIGHTOFFSET_STATE</th></tr>
<tr><td></td><td>0</td><td><b>RefPicListNum</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Reference Picture List 0</td></tr><tr><td>1</td><td>Reference Picture List 1</td></tr></table></td></tr>
<tr><td>2..17</td><td>511:0</td><td><b>LumaOffsets</b>
<table><tr><td>Format:</td><td>HCP_WEIGHTOFFSET_LUMA_ENTRY[16]</td></tr></table></td></tr>
<tr><td>18..33</td><td>511:0</td><td><b>ChromaOffsets</b>
<table><tr><td>Format:</td><td>HCP_WEIGHTOFFSET_CHROMA_ENTRY[16]</td></tr></table></td></tr>
<tr><td>34..41</td><td>255:0</td><td><b>ChromaOffsetsExt</b>
<table><tr><td>Format:</td><td>HCP_WEIGHTOFFSET_CHROMA_EXT_ENTRY[8]</td></tr></table></td></tr>
</table>

HEVC_SFC_AVS_CHROMA_Coeff_Table

HEVC_SFC_AVS_CHROMA_Coeff_Table		
Source: BSpec Length Bias: 2		
This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Default Value: 9h Media HEVC+SFC Mode Format: OpCode
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 6h SFC_AVS CHROMA Coeff_Table Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 3Fh Excludes DWord (0,1) Format: =n Total Length - 2
1..64	2047:0	**AVS CHROMA Coefficient Table Body** Format: SFC_AVS_CHROMA_COEFF_TABLE_BODY

HEVC_SFC_AVS_LUMA_Coeff_Table

HEVC_SFC_AVS_LUMA_Coeff_Table

Source:	BSpec
Length Bias:	2

This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Default Value: 9h Media HEVC+SFC Mode Format: OpCode
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 5h SFC_AVS LUMA Coeff_Table Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 7Fh Excludes DWord (0,1) Format: =n Total Length - 2
1..128	4095:0	**AVS LUMA Coefficient Table Body** Format: SFC_AVS_LUMA_COEFF_TABLE_BODY

HEVC_SFC_AVS_STATE

HEVC_SFC_AVS_STATE

Source:	BSpec
Length Bias:	2

This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Format: OpCode Value | Name 9h | Media HEVC+SFC Mode **[Default]**
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 2h SFC_AVS_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1..3	95:0	**AVS State Body** Format: SFC_AVS_STATE_BODY

HEVC_SFC_FRAME_START

HEVC_SFC_FRAME_START

Source:	BSpec
Length Bias:	2

This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Default Value: 9h Media HEVC+SFC Mode Format: OpCode
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 4h SFC_FRAME_START Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:0	**Frame Start Body** Format: SFC_FRAME_START_BODY

HEVC_SFC_IEF_STATE

HEVC_SFC_IEF_STATE		
Source:	BSpec	
Length Bias:	2	
This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Default Value: 9h Media HEVC+SFC Mode Format: OpCode
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 3h SFC_IEF_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 16h Excludes DWord (0,1) Format: =n Total Length - 2
1..23	735:0	**SFC IEF State Body** Format: SFC_IEF_STATE_BODY

HEVC_VP9_RDOQ_STATE

HEVC_VP9_RDOQ_STATE		
Source:	VideoCS	
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h Format: Opcode PARALLEL_VIDEO_PIPE
	28:27	**Pipeline** Default Value: 2h Format: Opcode MFX_COMMON
	26:23	**Opcode** Default Value: 7h Format: Opcode Codec/Engine Name = HCP = 7h
	22:21	**SubOpA** Default Value: 0h Format: Opcode
	20:16	**SubOpB** Default Value: 8h Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Total Length - 2 Value | Name | Description 98h | DWORD_COUNT_n **[Default]** | Excludes DWord (0,1)
1	31	**Disable HTQ performance fix0** set to disable performance optimizations done by doubling LNZ and OSR storage Set to 1, to go back to single LNZ and OSR (no optimization) Set to 0, to use double buffer LNZ and OSR
	30	**Disable HTQ performance fix1** set to disable performance optimization done to save 1clk while switching from EBB to GTRAM storage. This is critical for IntraLoop performance Set to 1, disable optimization Set to 0, enable optimization

HEVC_VP9_RDOQ_STATE		
	29:0	**Reserved** Access: RO Format: MBZ
2..33	1023:0	**IntraLumaLambda_QP0_63** Format: HEVC_VP9_RDOQ_LAMBDA_FIELDS[32]
34..65	1023:0	**IntraChromaLambda_QP0_63** Format: HEVC_VP9_RDOQ_LAMBDA_FIELDS[32]
66..97	1023:0	**InterLumaLambda_QP0_63** Format: HEVC_VP9_RDOQ_LAMBDA_FIELDS[32]
98..129	1023:0	**InterChromaLambda_QP0_63** Format: HEVC_VP9_RDOQ_LAMBDA_FIELDS[32]
130..135	191:0	**IntraLumaLambda_QP64_75** Format: HEVC_VP9_RDOQ_LAMBDA_FIELDS[6]
136..141	191:0	**IntraChromaLambda_QP64_75** Format: HEVC_VP9_RDOQ_LAMBDA_FIELDS[6]
142..147	191:0	**InterLumaLambda_QP64_75** Format: HEVC_VP9_RDOQ_LAMBDA_FIELDS[6]
148..153	191:0	**InterChromaLambda_QP64_75** Format: HEVC_VP9_RDOQ_LAMBDA_FIELDS[6]

HI8DS Render Target Write MSD

MSD_RTW_HI8DS - HI8DS Render Target Write MSD

Source:	EuSubFunctionRenderDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message
	29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Per-Coarse Pixel PS outputs enable** Format: Enable This bit indicates the render target write is a coarse pixel write. **Programming Notes** This bit must be DISABLED if a pixel shader thread is dispatched at pixel- or sample- rate. This bit may be DISABLED if a multi-rate pixel shader thread is dispatched at coarse rate. In such case, it indicates per-pixel or per-sample write (as determined by Per-Sample PS outputs enable) from a multi-rate shader, where the set of pixels is indicated by the Pixel shading phase field in Extended Message descriptor.
	17:14	**Message Type** Default Value: 0Ch Format: Opcode Render Target Write message

<table>
<tr><th colspan="3">MSD_RTW_HI8DS - HI8DS Render Target Write MSD</th></tr>
<tr><td></td><td>13</td><td><b>Per-Sample PS Enable</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot.
<table><tr><th>Programming Notes</th></tr>
<tr><td>This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE.
When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0.
When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples).
When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.</td></tr></table>
</td></tr>
<tr><td></td><td>12</td><td><b>Last Render Target Select</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.
</td></tr>
<tr><td></td><td>11</td><td><b>Slot Group Select</b>
<table><tr><td>Format:</td><td>MDC_RT_SGS</td></tr></table>
This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
</td></tr>
<tr><td></td><td>10:8</td><td><b>Render Target Message Subtype</b>
<table><tr><td>Default Value:</td><td>3h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table>
SIMD8 dual source message. Use slots [15:8] for pixel enables, X/Y addresses, and oMask.
<table><tr><th>Programming Notes</th></tr>
<tr><td>The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:24] are referenced instead of [15:8].</td></tr></table>
</td></tr>
<tr><td></td><td>7:0</td><td><b>Binding Table Index</b>
<table><tr><td>Format:</td><td>MDC_BTS</td></tr></table>
Specifies the Binding Table Index for the message
</td></tr>
</table>

Hword Aligned Block Read MSD

MSD0R_HWAB - Hword Aligned Block Read MSD

Source:	EuSubFunctionDataPort0
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 01h Format: Opcode Aligned Block Read message
	13	**Block Message Subtype** Default Value: 1 Format: Opcode Hword Block Read/Write subtype
	12:11	**Reserved** Access: RO Format: MBZ
	10:8	**Data Elements** Format: MDC_DB_HW Specifies the number of registers to be read

MSD0R_HWAB - Hword Aligned Block Read MSD		
	7:0	**Binding Table Index** Format: **MDC_BTS_SLM_A32** Specifies the Binding Table Index for the message

Hword Aligned Block Write MSD

MSD0W_HWAB - Hword Aligned Block Write MSD

Source:	EuSubFunctionDataPort0
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 09h Format: Opcode Aligned Block Write message
	13	**Block Message Subtype** Default Value: 1 Format: Opcode Hword Block Read/Write subtype
	12:11	**Reserved** Access: RO Format: MBZ
	10:8	**Data Elements** Format: MDC_DB_HW Specifies the number of registers to be written

MSD0W_HWAB - Hword Aligned Block Write MSD		
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

If

if - If

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

An if instruction starts an if/endif or an if/else/endif block of code. It restricts execution within the conditional block to only those channels that were enabled via the predicate control. Each if instruction must have a matching endif instruction and may have up to one matching else instruction before the matching endif. If all channels are inactive (for the if/endif or if/else/endif block), a jump is performed to the instruction referenced by JIP. This jump must be to right after the matching else instruction when present, or otherwise to the matching endif instruction of the conditional block. If SPF is ON, the UIP must be used to update IP; JIP is not used in this case.

The following table describes the 32-bit exit code <JIP> and <UIP>. If <branch_ctrl> is set, then the JIP points to the first join instruction within the if block. If <branch_ctrl> is not set, <JIP> should point to the instruction right after the matching else instruction if it exists, otherwise <JIP> should point to the endif instruction. <UIP> should always point to the endif instruction. When a jump occurs, this value is added to IP pre-increment. In instruction binary, <JIP> and <UIP> are at location <src0> & <src1> and must be of type D (signed dword integer).

Format:

```
[(pred)] if (exec_size JIP UIP <branch_ctrl>
```

Restriction
The execution size must be the same for the if, else, and endif instructions of the same code block.

Syntax

```
[(pred)] if (exec_size) imm32 imm32 <branch_ctrl>
```

Pseudocode

```
Evaluate(WrEn);
for ( n = 0; n < 32; n++ ) {
    if ( WrEn.channel[n] == 0 ) {
        PcIP[n] = IP + JIP;
    } else {
        PcIP[n] = IP + 1;
    }
}
if ( PcIP != (IP + 1) ) {  // for all channels
    Jump(IP + JIP);
}
```

DWord	Bit	Description

if - If

0..3	127:96	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	127:96	**JIP** Exists If: ([Src0.IsImm]==true) Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:80	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	95:64	**Reserved** Exists If: ([Src0.IsImm]==true) AND ([Src1.IsImm]==false) Format: MBZ
	95:64	**UIP** Exists If: ([Src0.IsImm]==true) AND ([Src1.IsImm]==true) Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	79:66	**Src0.Operand** Exists If: ([Src0.IsImm]==false) Format: DirectOperand
	65:64	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	63:50	**Dst.Operand** Format: DirectOperand
	49:48	**Reserved** Access: RO Format: MBZ
	47	**Src1.IsImm** This field indicate that Source 1 operand is carrying an immediate value Value: 0 — Name: false Value: 1 — Name: true
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value Value: 0 — Name: false

<table>
<tr><th colspan="3">if - If</th></tr>
<tr><td></td><td></td><td><table><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:34</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>BranchCtrl</b>
This field is used by <b><i>goto</i></b>, <b>if</b>, and <b><i>else</i></b> instructions to control branching. See the <b>goto</b> instruction description for more information about BranchCtrl.</td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">if - If</th></tr>
<tr><td></td><td></td><td><table><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td>PredCtrl</td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td>ChanOff</td></tr></table>This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td>ExecSize</td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td>Header</td></tr></table></td></tr>
</table>

Illegal

illegal - Illegal

Source:	EuIsa
Length Bias:	4
Predication:	false
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The Illegal Opcode Exception Enable flag in cr0.1 is normally set so the normal processing of an illegal opcode is to transfer control to the System Routine. Instruction dispatch treats any unused 8-bit opcode (including bit 7 of the instruction, reserved for future opcode expansion) as if it is the illegal opcode. The illegal opcode is zero because that byte value is more likely than most to be read via a wayward instruction pointer. The illegal instruction is an instruction only in the same way that a NULL pointer in software is a pointer. Both are special values indicating invalid instances.

Format:

```
illegal
```

Restriction

The illegal instruction takes no instruction options.

Syntax

```
illegal
```

Pseudocode

```
{
 Set the Illegal Opcode Exception Status bit in cr0.1.
     if ( Illegal Opcode Exception Enable is set in cr0.1 ) {
         Transfer control to the System Routine (return address to AIP, IP = SIP).
     }
 }
```

DWord	Bit	Description
0..3	127:7	**Reserved** Access: RO Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE

Join

join - Join

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The join instruction makes the inactive channels active at the join IP if those channels are predicated. Any deactivated channels due to a goto instruction match the join IP are activated (qualified with predicates at join). If no IP is matched at this join, the program goes to the next IP with the active channels which followed the program path up to the join instruction. If no active channels are present after executing the join instruction, the program jumps to the offset specified by JIP instead of next IP. The join instruction is used in conjunction with a goto instruction. The join activates channels that are deactivated by the goto instruction. See the goto instruction for the deactivation rules. The goto and join instructions enable unstructured program control flow. These instructions must be used with additional care where dangling channels can result without proper compiler checks, meaning that it is expected that programs will navigate through these paths to reactivate the channels. Hardware does not provide native checks or reconvergence. The following table describes the 32-bit JIP. In instruction binary, JIP is at location src1 and must be of type D (signed DWord integer). JIP must be an immediate operand and is a signed 32-bit number. This value is added to IP pre-increment. If SPF is ON, none of the PcIP are updated.

Format:

```
[(pred)] join (exec_size) JIP
```

Programming Notes

An index of 0 is an infinite loop.

Restriction

JIP must point to a Flow Control Instruction.

Syntax

```
[(pred)] join (exec_size) imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if (WrEn.chan[n] ) {  // for the predicated channels and the remaining channels
         PcIP[n] = IP + 1;
     }
 }
 if ( PcIP != (IP + 1) ) {  // for all channels when no channels are activated and no
other active channels
     Jump(IP + JIP);
 }
```

join - Join

DWord	Bit	Description
0..3	127:96	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	127:96	**JIP** Exists If: ([Src0.IsImm]==true) Format: S31 The byte-aligned jump distance if a jump is taken for the channel
	95:80	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	95:64	**Reserved** Exists If: ([Src0.IsImm]==true) Format: MBZ
	79:66	**Src0.Operand** Exists If: ([Src0.IsImm]==false) Format: **DirectOperand**
	65:64	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	63:50	**Dst.Operand** Format: **DirectOperand**
	49:47	**Reserved** Access: RO Format: MBZ
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value. Value | Name 0 | false 1 | true
	45:34	**Reserved** Access: RO Format: MBZ
	33	**BranchCtrl** This field is used by ***goto***, **if,** and ***else*** instructions to control branching. See the **goto** instruction description for more information about BranchCtrl.

<table>
<tr><th colspan="3">join - Join</th></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td>PredCtrl</td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
</table>

join - Join		
	23	**FlagRegNum[0]** This field specifies bit[0] of the register number for a flag register operand.
	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: ChanOff This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: ExecSize This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: Header

Jump Indexed

jmpi - Jump Indexed

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

Description

The jmpi instruction redirects program execution to an index offset relative to the pre-incremented instruction pointer. The index is a signed integer value, with positive or zero integers for forward jumps, and negative integers for backward jumps. In instruction binary, index is carried as src0 register or immediate. The ip register must be put (for example, by the assembler) at the dst. Predication is allowed to provide conditional jump with a scalar condition. As the execution size is 1, the first channel of PMASK (flags post prediction control and negate) is used to determine whether the jump is taken or not. If the condition is false, the jump is not taken and execution continues with the next instruction.

Format:

```
[(pred)] jmpi (1) index {NoMask}
```

Programming Notes

An index argument of 16 would continue to the next instruction (assuming the instruction is encoded as 128b).

An index argument of 0 loops infinitely: all immediate branch arguments, including jmpi now, are relative to the pre-increment IP.

Restriction

The execution size must be 1.

MaskCtrl must be specified.

QtrCtrl must not be used for jmpi instruction.

Jmpi instruction with non-uniform predicates or reg32 source cannot be used when EU Fusion is enabled.

Syntax

```
[(pred)] jmpi (1) reg32 {NoMask}
 [(pred)] jmpi (1) imm32 {NoMask}
```

Pseudocode

```
Evaluate(WrEn);
 if ( WrEn != 0 ) {
     Jump(IP + index );
 }
```

DWord	Bit	Description
0..3	127:96	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	127:96	**JIP** Exists If: ([Src0.IsImm]==true) Format: S31 The byte-aligned jump distance if a jump is taken for the channel
	95:80	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	95:64	**Reserved** Exists If: ([Src0.IsImm]==true) Format: MBZ
	79:66	**Src0.Operand** Exists If: ([Src0.IsImm]==false) Format: **DirectOperand**
	65:64	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	63:50	**Dst.Operand** Format: **DirectOperand**
	49:47	**Reserved** Access: RO Format: MBZ
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value. **Value** / **Name** 0 / false 1 / true
	45:34	**Reserved** Access: RO Format: MBZ
	33	**BranchCtrl** This field is used by ***goto***, **if**, and ***else*** instructions to control branching. See the **goto** instruction description for more information about BranchCtrl.
	32	**AtomicCtrl** Format: **AtomicCtrl**

<table>
<tr><th colspan="3">jmpi - Jump Indexed</th></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr>
<tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b>
<table>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td>PredCtrl</td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
</table>

jmpi - Jump Indexed

	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: **ChanOff** This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

L3_CONTROL

<table>
<tr><th colspan="3">L3_CONTROL - L3_CONTROL</th></tr>
<tr><td colspan="3">Source: BSpec
Length Bias: 2</td></tr>
<tr><td colspan="3">This command provides flexibility to flush selective graphics memory address locations (4KB in size) cached in L3$ without requiring to flush the complete L3$. The memory pages to be flushed must be detailed through L3 Flush Address Range record in line to the command. Multiple L3 Flush Address Range records can be programmed through a single L3_CONTROL command.
L3 Flush Address Range has number of pages to be flushed and must be a power of two (2^n), this is indicated in terms of number of lower order bits of the address to be masked (AddressMask). L3 Flush Address Range has the starting page of the graphics virtual address to be flushed and must be a power of two (2^m) with a greater value than that of the number of pages (m >=n). Refer L3 Flush Address Range structure for examples.
This command is supported by RenderCS and ComputeCS.</td></tr>
<tr><td colspan="3"><table><tr><th>Programming Notes</th></tr><tr><td>• SW must always program Post-Sync operation address and data qword fields in the command. Hardware will ignore these fields when Post-Sync Operation is not enabled in the command.</td></tr></table></td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Type</b><table><tr><td>Default Value:</td><td>03h GFXPIPE</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Command SubType</b><table><tr><td>Default Value:</td><td>3h GFXPIPE_3D</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><b>3D Command Opcode</b><table><tr><td>Default Value:</td><td>5h L3_CONTROL</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>23</td><td><b>3D Command Sub Opcode</b><table><tr><td>Default Value:</td><td>1h L3_CONTROL</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>22</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>21</td><td><b>Destination Address Type</b>
Defines address space of Destination Address<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>PPGTT</td><td>Use PPGTT address space for DW write</td></tr><tr><td>1h</td><td>GGTT</td><td>Use GGTT address space for DW write</td></tr></table></td></tr>
</table>

L3_CONTROL - L3_CONTROL

Programming Notes
Ignored if ""No Write" is selected in Post-Sync Operation.

20 **Command Streamer Stall Enable**

Format:	U1

If ENABLED, Command Parser stalls on this command until the command is completely executed.

19 **Reserved**

18:16 **Reserved**

Access:	RO
Format:	MBZ

15 **Post Sync Operation L3 Cacheability Control**

Value	Name	Description
0h	Default MOCS **[Default]**	MOCS value will be **CS Write Format Override**0x20c4[13:7]
1h	Cacheable MOCS	MOCS value will be **MOCS Index for Command Buffer Caching** 0x2084[6:0]

14 **Post Sync Operation**

This field specifies an optional action to be taken upon completion of the synchronization operation.

Value	Name	Description
0h	No Write	No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc.
1h	Write Immediate Data	Write the QWord containing Immediate Data Low, High DWs to the Destination Address

13 **Un-Typed Data-Port Cache Flush**

This bit controls the flushing of the data-port's Un-Typed data cache.

If set, dataport ensures all the dirty lines in un-typed data cache as flushed to memory and are coherent in L3 cache as part of the flush operation.

Programming Notes	Source
"HDC Pipeline Flush" bit must be set.	
This bit is functional and must be only set for GPGPU workloads, i.e when PIPELINE_SELECT command has set “Pipeline Select” mode set to “GPGPU”.	RenderCS

12 **PSS Stall Sync Enable**

If set, PSS Units will stall successive PS threads from being dispatched until all the prior PS threads complete. Once all PSSs are synced up (across Slices), L3 flush as per the address range takes place and on completion PSS units will get un-stalled.

L3_CONTROL - L3_CONTROL

	11	**Depth Stall Sync Enable** If set, 3D pipeline will stall any subsequent primitives at the Depth Test stage until they Sync across all the slices. Once all the Depth Test Stages are synced up (across Slices), L3 flush as per the address range takes place and on completion Depth Test Stages gets un-stalled.
	10	**HDC Pipeline Flush** Setting this bit ensures HDC pipeline is flushed and the memory transactions are coherent in L3$ as part of the flush operation prior to triggering of address based range flush to L3
	9	**Render Target Cache Flush Enable** Setting this bit enables flushing (i.e. writing back the dirty lines to memory and invalidating the tags) of Render Cache (L1) prior to triggering of address based range flush to L3.
	8	**Depth Cache Flush** Setting this bit enables flushing (i.e. writing back the dirty lines to memory and invalidating the tags) of depth related caches (HiZ cache, Stencil cache and depth cache) prior to triggering of address based range flush to L3.
	7:0	**DWord Length** Format: =n n = 2b+3 (where 'b' is number of L3 Flush Address Ranges) **Value** | **Name** [3,255] | 0 - 126 L3 Flush Address Ranges
1..2	63:48	**Reserved** Access: RO Format: MBZ
	47:3	**Address** Format: GraphicsAddress[47:3] Bits 47:3 specify the QW aligned graphics memory address to which the Immediate Qword Data is reported on execution of this command. Ignored if "No Write" is selected in Post-Sync Operation.
	2:0	**Reserved** Access: RO Format: MBZ
3..4	63:0	**Immediate Data** Format: U64 This field specifies the QWord value to be written to the address when Post-Sync Operation is enabled. Only valid when Post-Sync Operation is 1h (Write Immediate Data) and ignored if "No Write" is selected in Post-Sync Operation.
5..n	63:0	**L3_FLUSH_ADDRESS_RANGE** Format: L3_FLUSH_ADDRESS_RANGE

Leading Zero Detection

lzd - Leading Zero Detection

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

The lzd instruction counts component-wise the leading zeros from src0 and stores the resulting counts in dst. If src0 is zero, store 32 in dst.

Format:

```
[(pred)] lzd[.cmod] (exec_size) dst src0
```

Syntax

```
[(pred)] lzd[.cmod] (exec_size) reg reg
 [(pred)] lzd[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         UD udScalar = src0.chan[n];
         UD cnt = 0;
         while ( (udScalar & (1 « 31)) == 0 && cnt != 32 ) {
             cnt ++;
             udScalar = udScalar « 1;
         }
         dst.chan[n] = cnt;
     }
 }
```

Src Types	Dst Types
*B,*W,*D	UD

DWord	Bit	Description
0..3	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true)
	95:92	**CondCtrl** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **FlagModifier**
	95:64	**Src0.ImmValue[63:32]** Exists If: ([Src0.IsImm]==true) AND (([Src0.DataType]==:q) OR ([Src0.DataType]==:uq) OR ([Src0.DataType]==:df))

lzd - Leading Zero Detection

	87:84	**Src0.VertStride**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
			Format:	**VertStride**
	83:81	**Src0.Width**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
			Format:	**Width**
	80	**Src0.AddrMode**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
			Format:	**AddrMode**
	79:66	**Src0.Operand**	Exists If:	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Direct)
			Format:	**DirectOperand**
	79:66	**Src0.Operand**	Exists If:	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Indirect)
			Format:	**IndirectOperand**
	65:64	**Src0.HorzStride**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
			Format:	**HorzStride**
	63:50	**Dst.Operand**	Exists If:	([Dst.AddrMode]==Indirect)
			Format:	**IndirectOperand**
	63:50	**Dst.Operand**	Exists If:	([Dst.AddrMode]==Direct)
			Format:	**DirectOperand**
	49:48	**Dst.HorzStride**	Format:	**HorzStride**
	47	**Reserved**	Access:	RO
			Format:	MBZ
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value.		

<table>
<tr><th colspan="3">lzd - Leading Zero Detection</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td>SrcMod</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td>ImmDataType</td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.</td></tr>
</table>

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table>
</td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table>
</td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td><b>PredCtrl</b></td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table>
<tr><td>Format:</td><td><b>ChanOff</b></td></tr>
</table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table>
<tr><td>Format:</td><td><b>ExecSize</b></td></tr>
</table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b>
<table>
<tr><td>Format:</td><td><b>Header</b></td></tr>
</table>
</td></tr>
</table>

LO8DS Render Target Write MSD

MSD_RTW_LO8DS - LO8DS Render Target Write MSD

Source:	EuSubFunctionRenderDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message
	29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Per-Coarse Pixel PS outputs enable** Format: Enable This bit indicates the render target write is a coarse pixel write. **Programming Notes** This bit must be DISABLED if a pixel shader thread is dispatched at pixel- or sample- rate. This bit may be DISABLED if a multi-rate pixel shader thread is dispatched at coarse rate. In such case, it indicates per-pixel or per-sample write (as determined by Per-Sample PS outputs enable) from a multi-rate shader, where the set of pixels is indicated by the Pixel shading phase field in Extended Message descriptor.
	17:14	**Message Type** Default Value: 0Ch Format: Opcode Render Target Write message

MSD_RTW_LO8DS - LO8DS Render Target Write MSD

	13	**Per-Sample PS Enable** Format: | Enable If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot. **Programming Notes** This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Last Render Target Select** Format: | Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.
	11	**Slot Group Select** Format: | MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
	10:8	**Render Target Message Subtype** Default Value: | 2h Format: | Opcode SIMD8 dual source message. Use slots [7:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].
	7:0	**Binding Table Index** Format: | MDC_BTS Specifies the Binding Table Index for the message

Load

DP_LOAD - Load

Source:	SFID_1, SFID_6, SFID_E, SFID_F
Length Bias:	1

Load untyped data from memory. For each enabled SIMT lane, a vector is read from memory into registers.

Programming Notes
The src0 address payload format is selected by Address Size.
The dest data payload format is selected by Data Size. If not transposed, Vector Size specifies how many sequential copies of the data payload are in the message. If transposed, the Exec_Mask specifies how many sequential copies of the data payload are in the message.

Restriction	Source
Restriction : Load is not supported by data port URB.	SFID_6

Syntax

```
[(pred)] LOAD.sfid[.cache] (exec_mask) dest_reg:data_size[.vect_size][transpose]
<addr_type[+offset]>src0_reg:addr_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base;
for (n = 0; n < 32; n++) {
    if (Msg.ChEn[n]) {
        for (v = 0; v < vect_size; v++) {
            if (transpose) {
                dest[n].data_size[v] = ((msg_base_address+offset)+(src0.addr_size[n])).data_size[v]
            } else {
                dest[v].data_size[n] = ((msg_base_address+offset)+(src0.addr_size[n])).data_size[v]
            }
        }
    }
}
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful load messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful load messages to UGML (SFID_1) is only allowed for SURFTYPE_BUFFER. Load messages to SLM (SFID_E) and URB (SFID_6) must have DP_ADDR_TYPE as FLAT.

DP_LOAD - Load

28:25 **Src0 Length**

Format:	DP_ADDR_REG_SIZE

Specifies the size of the address payload, in registers.

Programming Notes
src0_length = roundup((addr_size * simd_size) / grf_size)) simd_size is 16, if transpose is 0. simd_size is 1 if transpose is 1.
src0_length = roundup((addr_size * simd_size) / grf_size)) simd_size is 8 or 16, if transpose is 0. simd_size is 1 if transpose is 1.

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-16		Data payload size, in registers.	dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 16, if transpose is 0. simd_size is 1 if transpose is 1. dest_length = roundup((data_size * vector_length * simd_size) / grf_size)) simd_size is 8 or 16, if transpose is 0. simd_size is 1 if transpose is 1.
0		Pre-fetch into data cache. No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_LOAD

Specifies how the instruction overrides the cache settings.

16 **Reserved**

Access:	RO
Format:	MBZ

15 **Transpose**

Format:	DP_TRANSPOSE

Specifies if the registers are a transposed data vector.

Restriction
Transposed vectors are restricted to Exec_Mask == 1.

14:12 **Vector Size**

Format:	DP_VECT_SIZE

Specifies the vector length of each data payload item.

DP_LOAD - Load

Programming Notes	Source
For dataport UGML (SFID_1), if DP_TRANSPOSE is Off, maximum vector size supported is 4. Moreover, vector size of 3 is not supported.	SFID_1

Restriction	Source
Loads with vector size of 8 or more is restricted to EXEC_MASK <= 8 (lower 8 lanes). Address payload format in this case is either A16_PAYLOAD_SIMT8, A32_PAYLOAD_SIMT8 or A64_PAYLOAD_SIMT8, depending on DP_ADDR_SIZE.	
Loads with data size of d64 and vector size of 3 or more is restricted to EXEC_MASK <= 8 (lower 8 lanes). Address payload format in this case is either A16_PAYLOAD_SIMT8, A32_PAYLOAD_SIMT8 or A64_PAYLOAD_SIMT8, depending on DP_ADDR_SIZE.	
Restriction: For dataports UGM, SLM, and URB, if DP_TRANSPOSE is Off, maximum vector size supported is 8 for D32, and 4 for D64.	SFID_6, SFID_E, SFID_F

11:9 **Data Size**

Format:	DP_DATA_SIZE

Specifies both bit size of the data payload item in memory and the bit size used in the register payload.

Restriction	Source
8b and 16b data sizes are only supported with vector size 1 and Transpose off.	
For UGML, data size of D64 is only allowed when Address size is A64. Also, when data size is D64, per-lane addresses must be QW aligned.	SFID_1

8:7 **Address Size**

Format:	DP_ADDR_SIZE

Specifies the bit size of each address payload item.

Restriction
If DP_VECT_SIZE is 1, the addresses can be byte aligned. If DP_VECT_SIZE is greater than 1, addresses must be aligned to data size.

6 **Reserved**

Access:	RO
Format:	MBZ

5:0 **Load Operation**

Default Value:	0 Load
Format:	Opcode

Load Cmask

DP_LOAD_CMASK - Load Cmask

Source:	SFID_1, SFID_D, SFID_E, SFID_F
Length Bias:	1

Load untyped data from memory. For each enabled SIMT lane and enabled component mask, a scalar is read from memory into registers.

Programming Notes
The src0 address payload format is selected by Address Size.
The dest data payload format is selected by Data Size. Cmask specifies how many sequential copies of the data payload are in the message.

Restriction	Source
This message is not supported for SFID_D (TGM).	
Restriction : Load_cmask is not supported by data port URB.	SFID_6

Syntax

```
[(pred)] LOAD_CMASK.sfid[.cache] (exec_mask) dest_reg:data_size[.cmask]
<addr_type[+offset]>src0_reg:addr_size
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base;
for (n = 0; n < 32; n++) {
    if (Msg.ChEn[n]) {
        for (m = v = 0; v < 4; v++) {
            if (cmask[v]) {
                dest[m].data_size[n] = ((msg_base_address+offset)+(src0.addr_size[n])).data_size[v];
                m++;
            }
        }
    }
}
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful load_cmask messages to UGM (SFID_F) is only allowed on SURFTYPE_BUFFER, SURFTYPE_SCRATCH and SURFTYPE_NULL. Stateful load_cmask messages to UGML (SFID_1) is only allowed on SURFTYPE_BUFFER. Load_cmask messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_LOAD_CMASK - Load Cmask

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinatesis the number of address coordinates used in the message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinatesis the number of address coordinates used in the message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description	Programming Notes
1-16		Data payload size, in registers.	dest_length = roundup((data_size * num_valid_channels(cmask)* simd_size) / grf_size)) simd_size is 16 dest_length = roundup((data_size * num_valid_channels(cmask)* simd_size) / grf_size)) simd_size is 8 or 16
0		Pre-fetch into data cache. No data returned in registers.	

19:17 **Cache**

Format:	DP_CACHE_LOAD

Specifies how the instruction overrides the cache settings.

16 **Reserved**

Access:	RO
Format:	MBZ

15:12 **Component Mask**

Format:	DP_CMASK

Specifies the component mask of each data payload item.

11:9 **Data Size**

Format:	DP_DATA_SIZE

Specifies both bit size of the data payload item in memory and the bit size used in the register payload.

Restriction
Only D32 is supported.

DP_LOAD_CMASK - Load Cmask		
	8:7	**Address Size** Format: DP_ADDR_SIZE Specifies the bit size of each address payload item.
	6	**Reserved** Access: RO Format: MBZ
	5:0	**Load Operation** Default Value: 2 Load Cmask Format: Opcode

Load Status

DP_LOAD_STATUS - Load Status

Source:	SFID_D, SFID_F
Length Bias:	1

Pre-fetch untyped data from memory. Returns status register, one bit per SIMT lane, indicating whether that lane was enabled and the address was in-bounds of the TRTT.

Programming Notes
The src0 address payload format is selected by Address Size.
The dest data payload is one register with the status bits (DP_STATUS_PAYLOAD).

Restriction
This message is not supported for SFID_D (TGM).

Syntax

```
[(pred)] LOAD_STATUS.sfid[.cache] (exec_mask) dest_reg:data_size[.vect_size][transpose]
<addr_type[+offset]>src0_reg:addr_size
```

Pseudocode

```
for (n = 0; n < 32; n++) { s = 1; if (Msg.ChEn[n]) { for (v = 0; v < vect_size;
v++) { if (s) { s = IsTRTT_VALID_PAGE(
((Base+offset)+(src0.addr_size[n])).data_size[v]); } } } dest.bit[n] = MsgChEn[n]
& s; }
```

```
for (n = 0; n < 16; n++) { s = 1; if (Msg.ChEn[n]) { for (v = 0; v < vect_size;
v++) { if (s) { s = IsTRTT_VALID_PAGE(
((Base+offset)+(src0.addr_size[n])).data_size[v]); } } } dest.bit[n] = MsgChEn[n]
& s; }
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful load_status messages to UGM (SFID_F) and UGML (SFID_1) areonly allowed on SURFTYPE_BUFFER.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_LOAD_STATUS - Load Status

Programming Notes
src0_length = roundup((addr_size * simd_size) / grf_size)) simd_size is 16 if transpose is 0. simd_size is 1 if transpose is 1.

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description
1		Pre-fetch into data cache. Status is returned in register.

19:17 **Cache**

Format:	DP_CACHE_LOAD

Specifies how the instruction overrides the cache settings.

16 **Reserved**

Access:	RO
Format:	MBZ

15 **Transpose**

Format:	DP_TRANSPOSE

Specifies if the registers are a transposed data vector.

Value	Name
0	Off

Restriction
Transposed vectors are restricted to Exec_Mask == 1 and Vector_size greater than 1.

14:12 **Vector Size**

Format:	DP_VECT_SIZE

Specifies the vector length of each data payload item.

Restriction	Source
Loads with vector size of 8 or more is restricted to EXEC_MASK <= 8 (lower 8 lanes).Address payload format in this case is either A16_PAYLOAD_SIMT8, A32_PAYLOAD_SIMT8 or A64_PAYLOAD_SIMT8, depending on DP_ADDR_SIZE.	
Loads with data size of d64 and vector size of 3 or more is restricted to EXEC_MASK <= 8 (lower 8 lanes).Address payload format in this case is either A16_PAYLOAD_SIMT8, A32_PAYLOAD_SIMT8 or A64_PAYLOAD_SIMT8, depending on DP_ADDR_SIZE.	
Restriction : For dataports UGM and SLM, if DP_TRANSPOSE is Off, maximum vector size supported is 8.	SFID_E, SFID_F

<table>
<tr><th colspan="3">DP_LOAD_STATUS - Load Status</th></tr>
<tr><td></td><td>11:9</td><td><b>Data Size</b>
<table><tr><td>Format:</td><td><b>DP_DATA_SIZE</b></td></tr></table>
Specifies both bit size of the data payload item in memory and the bit size used in the register payload.
<table><tr><th>Restriction</th></tr><tr><td>8b and 16b data sizes are only supported with vector size 1 and Transpose off.</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b>
<table><tr><td>Format:</td><td><b>DP_ADDR_SIZE</b></td></tr></table>
Specifies the bit size of each address payload item.
<table><tr><th>Restriction</th></tr><tr><td>If DP_VECT_SIZE is 1, the addresses can be byte aligned. If DP_VECT_SIZE is greater than 1, addresses must be aligned to data size.</td></tr></table></td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Load Operation</b>
<table><tr><td>Default Value:</td><td>27 Load Status</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Logic And

and - Logic And

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	false
Source Modifier:	true

The and instruction performs component-wise logic AND operation between src0 and src1 and stores the results in dst. Register source operands can use source modifiers: Any source modifier is logical, optionally changing a source value s to ~s (inverting all source bits). This capability allows expressions like a AND (NOT b) to be calculated with one instruction. This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers should be used.

Format:

```
      Source modifier is not allowed if source is an accumulator.
```

Restriction
Source modifier is not allowed if source is an accumulator.

Syntax

```
[(pred)] and[.cmod] (exec_size) reg reg reg
 [(pred)] and[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = src0.chan[n] & src1.chan[n];
     }
 }
```

Src Types	Dst Types
*B,*W,*D	*B,*W,*D

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)
	125:122	**Reserved** Exists If: ([Src1.IsImm]==false)

and - Logic And

		Format:	MBZ
121:120	**Src1.Mod**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**SrcMod**
119:116	**Src1.VertStride**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**VertStride**
115:113	**Src1.Width**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**Width**
112	**Src1.AddrMode**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**AddrMode**
111:98	**Src1.Operand**		
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
		Format:	**IndirectOperand**
111:98	**Src1.Operand**		
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
		Format:	**DirectOperand**
97:96	**Src1.HorzStride**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**HorzStride**
95:92	**CondCtrl**		
		Format:	**FlagModifier**
91:88	**Src1.DataType**		
		Exists If:	([Src1.IsImm]==true)
		Format:	**ImmDataType**
91:88	**Src1.DataType**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**RegDataType**
87:84	**Src0.VertStride**		
		Format:	**VertStride**
83:81	**Src0.Width**		
		Format:	**Width**
80	**Src0.AddrMode**		
		Format:	**AddrMode**

<table>
<tr><th colspan="3">and - Logic And</th></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src1.IsImm</b>
This field indicate that Source 1 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
</table>

and - Logic And

<table>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr></table></td></tr>
</table>

and - Logic And

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td><b>Header</b></td></tr></table></td></tr>
</table>

Logic Not

not - Logic Not

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	false
Source Modifier:	true

The not instruction performs logical NOT operation (or one's complement) of src0 and storing the results in dst. This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers should be used.

A register source operand can use a source modifier: Any source modifier is logical, optionally changing a source value s to ~s (inverting all source bits). Such a source modifier is not particularly useful with the not instruction, as it changes the effect of not to just copying bits.

Format:

```
[(pred)] not[.cmod] (exec_size) dst src0
```

Restriction
Source modifier is not allowed if source is an accumulator.

Syntax

```
[(pred)] not[.cmod] (exec_size) reg reg
 [(pred)] not[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = ~ src0.chan[n];
     }
 }
```

Src Types	Dst Types
*B,*W,*D	*B,*W,*D

DWord	Bit	Description
0..3	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true)
	95:92	**CondCtrl** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **FlagModifier**

not - Logic Not

	95:64	**Src0.ImmValue[63:32]**	Exists If:	([Src0.IsImm]==true) AND (([Src0.DataType]==:q) OR ([Src0.DataType]==:uq) OR ([Src0.DataType]==:df))
	87:84	**Src0.VertStride**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
			Format:	**VertStride**
	83:81	**Src0.Width**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
			Format:	**Width**
	80	**Src0.AddrMode**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
			Format:	**AddrMode**
	79:66	**Src0.Operand**	Exists If:	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Direct)
			Format:	**DirectOperand**
	79:66	**Src0.Operand**	Exists If:	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Indirect)
			Format:	**IndirectOperand**
	65:64	**Src0.HorzStride**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
			Format:	**HorzStride**
	63:50	**Dst.Operand**	Exists If:	([Dst.AddrMode]==Indirect)
			Format:	**IndirectOperand**
	63:50	**Dst.Operand**	Exists If:	([Dst.AddrMode]==Direct)
			Format:	**DirectOperand**
	49:48	**Dst.HorzStride**	Format:	**HorzStride**

not - Logic Not

<table>
<tr><td></td><td>47</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td>SrcMod</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td>ImmDataType</td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">not - Logic Not</th></tr>
<tr><td></td><td></td><td>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td><b>PredCtrl</b></td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table>
<tr><td>Format:</td><td><b>ChanOff</b></td></tr>
</table>
This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
</table>

not - Logic Not		
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Logic Or

or - Logic Or

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	false
Source Modifier:	true

The or instruction performs component-wise logic OR operation between src0 and src1 and stores the results in dst. This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers should be used.

Register source operands can use source modifiers: Any source modifier is logical, optionally changing a source value s to ~s (inverting all source bits). This capability allows expressions like a OR (NOT b) to be calculated with one instruction.

Format:

```
        [(pred)] or[.cmod] (exec_size) dst src0 src1
```

Restriction
Source modifier is not allowed if source is an accumulator.

Syntax

```
[(pred)] or[.cmod] (exec_size) reg reg reg
 [(pred)] or[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = src0.chan[n] | src1.chan[n];
     }
 }
```

Src Types	Dst Types
*B,*W,*D	*B,*W,*D

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)

or - Logic Or

125:122	**Reserved**	
	Exists If:	([Src1.IsImm]==false)
	Format:	MBZ
121:120	**Src1.Mod**	
	Exists If:	([Src1.IsImm]==false)
	Format:	**SrcMod**
119:116	**Src1.VertStride**	
	Exists If:	([Src1.IsImm]==false)
	Format:	**VertStride**
115:113	**Src1.Width**	
	Exists If:	([Src1.IsImm]==false)
	Format:	**Width**
112	**Src1.AddrMode**	
	Exists If:	([Src1.IsImm]==false)
	Format:	**AddrMode**
111:98	**Src1.Operand**	
	Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
	Format:	**IndirectOperand**
111:98	**Src1.Operand**	
	Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
	Format:	**DirectOperand**
97:96	**Src1.HorzStride**	
	Exists If:	([Src1.IsImm]==false)
	Format:	**HorzStride**
95:92	**CondCtrl**	
	Format:	**FlagModifier**
91:88	**Src1.DataType**	
	Exists If:	([Src1.IsImm]==true)
	Format:	**ImmDataType**
91:88	**Src1.DataType**	
	Exists If:	([Src1.IsImm]==false)
	Format:	**RegDataType**
87:84	**Src0.VertStride**	
	Format:	**VertStride**
83:81	**Src0.Width**	
	Format:	**Width**

or - Logic Or

<table>
<tr><td></td><td>80</td><td><b>Src0.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src1.IsImm</b>
This field indicate that Source 1 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">or - Logic Or</th></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields</td></tr>
</table>

<table>
<tr><th colspan="3">or - Logic Or</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td>PredCtrl</td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td>ChanOff</td></tr></table>This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td>ExecSize</td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td>Header</td></tr></table></td></tr>
</table>

Logic Xor

xor - Logic Xor

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	false
Source Modifier:	true

The xor instruction performs component-wise logic XOR operation between src0 and src1 and stores the results in dst. This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers should be used.

Register source operands can use source modifiers: Any source modifier is logical, optionally changing a source value s to ~s (inverting all source bits). This capability allows expressions like a XOR (NOT b) to be calculated with one instruction.

Format:

```
        [(pred)] xor[.cmod] (exec_size) dst src0 src1
```

Restriction

Source modifier is not allowed if source is an accumulator.

Syntax

```
[(pred)] xor[.cmod] (exec_size) reg reg reg
 [(pred)] xor[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = src0.chan[n] ^ src1.chan[n];
     }
 }
```

Src Types	Dst Types
*B,*W,*D	*B,*W,*D

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)

xor - Logic Xor

125:122	**Reserved**	
	Exists If:	([Src1.IsImm]==false)
	Format:	MBZ
121:120	**Src1.Mod**	
	Exists If:	([Src1.IsImm]==false)
	Format:	**SrcMod**
119:116	**Src1.VertStride**	
	Exists If:	([Src1.IsImm]==false)
	Format:	**VertStride**
115:113	**Src1.Width**	
	Exists If:	([Src1.IsImm]==false)
	Format:	**Width**
112	**Src1.AddrMode**	
	Exists If:	([Src1.IsImm]==false)
	Format:	**AddrMode**
111:98	**Src1.Operand**	
	Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
	Format:	**IndirectOperand**
111:98	**Src1.Operand**	
	Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
	Format:	**DirectOperand**
97:96	**Src1.HorzStride**	
	Exists If:	([Src1.IsImm]==false)
	Format:	**HorzStride**
95:92	**CondCtrl**	
	Format:	**FlagModifier**
91:88	**Src1.DataType**	
	Exists If:	([Src1.IsImm]==true)
	Format:	**ImmDataType**
91:88	**Src1.DataType**	
	Exists If:	([Src1.IsImm]==false)
	Format:	**RegDataType**
87:84	**Src0.VertStride**	
	Format:	**VertStride**
83:81	**Src0.Width**	
	Format:	**Width**

xor - Logic Xor

<table>
<tr><td></td><td>80</td><td><b>Src0.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src1.IsImm</b>
This field indicate that Source 1 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
</table>

xor - Logic Xor

<table>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td><b>Saturate</b></td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td><b>AccWrCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields</td></tr>
</table>

xor - Logic Xor

<table>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td><b>Header</b></td></tr></table></td></tr>
</table>

Media Block Read MSD

MSD1R_MB - Media Block Read MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 04h Format: Opcode Media Block Read message
	13:11	**Reserved** Access: RO Format: MBZ
	10:8	**Vertical Line Stride Override** Format: MDC_VLSO If enabled, specifies the Vertical Line Stride and Vertical Line Stride Offset override fields.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Media Block Write MSD

MSD1W_MB - Media Block Write MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 0Ah Format: Opcode Media Block Write message
	13:11	**Reserved** Access: RO Format: MBZ
	10:8	**Vertical Line Stride Override** Format: MDC_VLSO If enabled, specifies the Vertical Line Stride and Vertical Line Stride Offset override fields.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Media Transpose Read MSD

MSD1R_TT - Media Transpose Read MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 00h Format: Opcode Transpose Read message
	13:8	**Reserved** Access: RO Format: MBZ
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Memory Fence MSD

<table>
<tr><th colspan="3">MSD_MEMFENCE - Memory Fence MSD</th></tr>
<tr><td colspan="3">Source: EuSubFunctionDataPort0
Length Bias: 1</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>28:25</td><td><b>Message Length</b><table><tr><td>Format:</td><td>U4</td></tr></table>Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.</td></tr>
<tr><td></td><td>24:20</td><td><b>Response Length</b><table><tr><td>Format:</td><td>U5</td></tr></table>Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.</td></tr>
<tr><td></td><td>19</td><td><b>Header Present</b><table><tr><td>Format:</td><td>MDC_MHP</td></tr></table>Indicates that the message requires a header.</td></tr>
<tr><td></td><td>18</td><td><b>Legacy Message</b><table><tr><td>Default Value:</td><td>0h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table>Legacy Message</td></tr>
<tr><td></td><td>17:14</td><td><b>Message Type</b><table><tr><td>Default Value:</td><td>07h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table>Memory Fence message</td></tr>
<tr><td></td><td>13</td><td><b>Commit</b><table><tr><td>Format:</td><td>Enable</td></tr></table>Specifies whether control is returned to the thread only after the fence has been honored.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>1</td><td>Enabled <b>[Default]</b></td><td>The commit writeback register is always required to guarantee ordering.</td></tr><tr><td>0</td><td>Reserved</td><td>The commit writeback register is always required to guarantee ordering.</td></tr></table></td></tr>
<tr><td></td><td>12:9</td><td><b>L3 Flush</b>
The L3 Flush control is one of the following GSYNC signals.</td></tr>
</table>

MSD_MEMFENCE - Memory Fence MSD

Value	Name	Description
0h	Disabled **[Default]**	The L3 caches are not flushed.
08h	RW Data	Causes the L3 to flush any RW data.
04h	Constant Data	Causes the L3 to invalidate any Constant data.
02h	Texture Data	Causes the L3 to invalidate any Texture data.
01h	Instructions	Causes the L3 to invalidate all GPU instruction caches.

Programming Notes
If multiple caches need to be flushed, the commands need to be sent separately.
When the memory fence completes, the GSYNC has been started, but may not yet be completed. To know when the GSYNC is completed, Issue any read to the L3 cache after an L3 Flush operation and wait for that data to be returned.

8 **L1 Flush Data**

Format:	Enable

When set, invalidate this subslice's L1 read-only data cache.

Restriction
When "L1 Flush Data" is set, the "L3 Flush" field must be set to 0. If both L1 and L3 needs to be invalidated/flushed, SW need to send two separate fence messages.

7:0 **Binding Table Index**

Format:	MDC_BTS_SLM_A32

Specifies whether global memory or shared local memory (SLM) is fenced with this operation.

Value	Name	Description
0FEh	SLM	Only SLM is fenced with this operation. Global memory is not fenced. Restriction : The L3 Flush and L1 Flush fields are ignored when SLM memory is selected.
00h	Global Memory **[Default]**	Only global memory is fenced with this operation. SLM memory is not fenced. Performance : When a program needs to guarantee that all global memory writes are globally observable before a thread retires, a Memory Fence operation is used immediately before the EOT message.

MFC_AVC_PAK_OBJECT

MFC_AVC_PAK_OBJECT

Source:	VideoCS
Length Bias:	2

The MFC_AVC_PAK_OBJECT command is the second primitive command for the AVC Encoding Pipeline. The same command is used for both CABAC and CAVLC modes. The MV Data portion of the bitstream is loaded as indirect data object. Before issuing a MFC_AVC_PAK_OBJECT command, all AVC MFX states need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of this command. MB record must be consecutive with no gaps, hence we do not need MB(x,y) in each MB command. Internal counter will keep track of the current MB address, starting from the Start_MB_In_Slice loaded at the beginning of each slice. MFC_AVC_PAK_OBJECT command follows the MbType definition like MFD. Many fields in this command are identical to that in VME output. This is intended to reduce software converting overhead from VME to PAK. Encoding statistical data such as the total size of the output bitstream are provided through MMIO registers. Software may access these registers through MI_STORE_REGISTER_MEM command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_AVC_PAK_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_ENC Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 000Ah DWORD_COUNT_n Format: =n

MFC_AVC_PAK_OBJECT		
1	31:10	**Reserved** Access: RO Format: MBZ
	9:0	**Indirect PAK-MV Data Length** This field provides the length in bytes of the indirect data, which contains all the MVs for the current MB (in any partitioning and subpartitioning form). A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect PAK-MV Data Start Address field is ignored. This field must have the same alignment as the Indirect PAK-MV Data Start Address. This field must be DW aligned (since each MV is 4 bytes in size).Driver has to derived this field from MV size (MV quantity in DXVA, exact size) *4 bytes per MV.
2	31:29	**Reserved** Access: RO Format: MBZ
	28:0	**Indirect PAK-MV Data Start Address Offset** This field specifies the memory starting address (offset) of the MV data to be fetched into PAK Subsystem for processing. This pointer is relative to the MFC Indirect PAK-MV Object Base Address. Hardware ignores this field if indirect data is not present, i.e. the Indirect PAK-MV Data Length is set to 0. It is a Dword aligned address in all AVC encoding configuration, since each MV is 4 bytes in size. **Value** / **Name** [0,512MB) /
3..10	255:0	**Inline Data** Format: U32[8] All the required MB level controls and parameters for encoding are captured as inline data of the MFC_AVC_PAK_OBJECT command. It has a fixed size of 8 DWs. Its definition is described in the next section.
11	31:0	**Reserved** Access: RO Format: MBZ
12..23	383:0	**Reserved**

MFC_JPEG_HUFF_TABLE_STATE

MFC_JPEG_HUFF_TABLE_STATE

Source:	VideoCS
Length Bias:	2

This Huffman table commands contains both DC and AC tables for either luma or chroma. Once a Huffman table has been defined for a particular destination, it replaces the previous tables stored in that destination and shall be used in the remaining Scans of the current image. Two Huffman tables for luma and chroma will be sent to H/W, and chroma table is used for both U and V.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_JPEG_HUFF_TABLE_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 7h JPEG Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Common Format: OpCode
	20:16	**SubOpcode B** Default Value: 3h MEDIA_ Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 0AEh Excludes DWord (0,1) Format: =n
1	31:1	**Reserved** Access: RO Format: MBZ
	0	**Huff Table ID** Format: U1 Huffman table destination identifier will specify one of two destinations at the encoder into which the Huffman table must be stored.

<table>
<tr><th colspan="3">MFC_JPEG_HUFF_TABLE_STATE</th></tr>
<tr><td></td><td></td><td><table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td></td><td>Huffman table 0</td></tr>
<tr><td>1</td><td></td><td>Huffman table 1</td></tr>
</table></td></tr>
<tr><td>2..13</td><td>383:0</td><td><b>DC_TABLE</b>
12 categories with code length and code word. Each run/size has 1-byte code length, and 2-byte code word.</td></tr>
<tr><td>14..175</td><td>5183:0</td><td><b>AC_TABLE</b>
162 run/size with code length and code word. Each run/size has 1-byte code length, and 2-byte code word.</td></tr>
</table>

MFC_JPEG_SCAN_OBJECT

MFC_JPEG_SCAN_OBJECT				
Source:	VideoCS			
Length Bias:	2			
Encoder Only				
DWord	**Bit**	**Description**		
0	31:29	**Command Type**		
			Default Value:	3h PARALLEL_VIDEO_PIPE
			Format:	OpCode
	28:27	**Pipeline**		
			Default Value:	2h MFC_JPEG_SCAN_OBJECT
			Format:	OpCode
	26:24	**Media Command Opcode**		
			Default Value:	7h JPEG_ENC
			Format:	OpCode
	23:21	**SubOpcode A**		
			Default Value:	2h
			Format:	OpCode
	20:16	**SubOpcode B**		
			Default Value:	9h
			Format:	OpCode
	15:12	**Reserved**		
			Access:	RO
			Format:	MBZ
	11:0	**DWord Length**		
			Default Value:	001h Excludes DWord (0,1)
			Format:	=n
1	31:26	**Reserved**		
			Access:	RO
			Format:	MBZ
	25:0	**MCU Count**		
			Format:	U26
		This field indicates the number of MCUs in the Scan. MCU Count = $M_x X M_y$ The number of MCUs in a row: $M_x= (X + (H_1 *8 -1)) / (H_1 *8)$The number of MCUs in a column: $M_y= (Y + (V_1*8 -1)) / (V_1*8)$*X*: The number of samples per line in Y-image *Y*: The number of lines in Y-image*H1*: Horizontal sampling factor of Y-image in the Frame header*V1*: Vertical sampling factor of Y-image in the Frame header		

MFC_JPEG_SCAN_OBJECT

2

31:25 **Reserved**

Access:	RO
Format:	MBZ

24:22 **Huffman AC Table**

AC Huffman table destination selector specifies one of two possible AC table destinations for each Y, U, V, or R, G, B. The AC Huffman tables must have been loaded in destination 0 and 1 by the time of issuing MFC_JPEG_HUFF_TABLE_STATE Command.

If AC table 0 is used for Y and AC table 1 is used for U and V, it will be set to 110b. If AC table 0 is used for R, G, and B, it will be set to 000b and so on. Refer to the table below for the summary of actions.

Value	Name	Description
0XXb	Bit24 (V0)	The third image component must use the AC table 0.
1XXb	Bit24 (V1)	The third image component must use the AC table 1.
X0Xb	Bit23 (U0)	The second image component must use the AC table 0.
X1Xb	Bit23 (U1)	The second image component must use the AC table 1.
XX0b	Bit22 (Y0)	The first image component must use the AC table 0.
XX1b	Bit22 (Y1)	The first image component must use the AC table 1.

Restriction
When InputSurfaceFormatYUV = RGB, because the order of input image components can be RGB, GBR, BGR,\ or YUV, **Bit22** is used for the first image component, **Bit23** is used for the second image component, and **Bit24** is used for the third image component.

21 **Reserved**

Access:	RO
Format:	MBZ

20:18 **Huffman DC Table**

DC Huffman table destination selector specifies one of two possible DC table destinations for each Y, U, V, or R, G, B. The DC Huffman tables shall have been loaded in destination 0 and 1 by the time of issuing MFC_JPEG_HUFF_TABLE_STATE Command.

if DC table 0 is used for Y and DC table 1 is used for U and V, it will be set to 110b. If DC table 0 is used for R, G, and B, it will be set to 000b and so on. Refer to the table below for the summary of actions.

Value	Name	Description
0XXb	Bit20 (V0)	The third image component must use the DC table 0.
1XXb	Bit20 (V1)	The third image component must use the DC table 1.
X0Xb	Bit19 (U0)	The second image component must use the DC table 0.
X1Xb	Bit19 (U1)	The second image component must use the DC table 1.

MFC_JPEG_SCAN_OBJECT

XX0b	Bit18 (Y0)	The first image component must use the DC table 0.
XX1b	Bit18 (Y1)	The first image component must use the DC table 1.

Restriction
When InputSurfaceFormatYUV = RGB, because the order of input image components can be RGB, GBR, BGR, YUV, **Bit18** is used for the first image component, **Bit19** is used for the second image component, and **Bit20** is used for the third image component.

17

Head Present Flag

If this flag is set to 0, then no MFC_JPEG_PAK_INSERT_OBJECT commands will be sent. If this flag is set to 1, then one or more MFC_JPEG_PAK_INSERT_OBJECT commands will be sent after MFC_JPEG_SCAN_OBJECT command.

Value	Name	Description
0		No insertion into the output bitstream buffer before Scan encoded bitstream
1		Headers, tables, App data insertion into the output bitstream buffer. HW will insert the insertion data before the Scan encoded bitstream.

16

Is Last Scan

If this flag is set, then HW will insert EOI (0xFFD9) to the end of Scan encoded bitstream.

Value	Name	Description
0		Not the last Scan.
1		Indicates that the current Scan is the last one.

15:0

Restart Interval

Format:	U16

Specifies the number of MCUs in an ECS, except for the last ECS. Restart maker is inserted periodically and it separates the two neighboring ECSs.

Value	Name
0-FFFFh	

Programming Notes
A value of '0' implies that the Scan Data has a single ECS.

MFC_MPEG2_PAK_OBJECT

MFC_MPEG2_PAK_OBJECT

Source:	VideoCS
Length Bias:	2

The MFC_MPEG2_PAK_OBJECT command is the second primitive command for the MPEG-2 Encoding Pipeline.
Different from AVC, the MV Data portion of the bitstream is loaded as part of MB control data.
Before issuing a MFC_MPEG2_PAK_OBJECT command, all MPEG2_MFX states need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of this command.
MB record must be consecutive with no gaps, hence we do not need MB(x,y) in each MB command. Internal counter will keep track of the current MB address, starting from the Start_MB_In_Slice loaded at the beginning of each slice.
MFC_ MPEG2_PAK_OBJECT command follows the MbType definition like MFD.
Encoding statistical data such as the total size of the output bitstream are provided through MMIO registers. Software may access these registers through MI_STORE_REGISTER_MEM command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_AVC_PAK_INSERT_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 3h MPEG2 Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h ENC Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h MEDIA_ Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 0007h Excludes DWord (0,1) Format: =n

MFC_MPEG2_PAK_OBJECT		
1..8	255:0	**Inline Data** Format: U32[8] All the required MB level controls and parameters for encoding are captured as inline data of the MFC_MPEG2_PAK_OBJECT command. It has a fixed size of 8 DWs. Its definition is described in the next section

MFC_MPEG2_SLICEGROUP_STATE

MFC_MPEG2_SLICEGROUP_STATE

Source:	VideoCS
Length Bias:	2

This is a slice group level command and can be issued multiple times within a picture that is comprised of multiple slice groups. The same command is used for AVC encoder (PAK mode) and decoder (VLD and IT modes).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MPEG2_SLICEGROUP_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 3h MPEG2 Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h MEDIA_ Format: OpCode
	20:16	**SubOpcode B** Default Value: 3h MEDIA_ Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 6h Excludes DWord (0,1) Format: =n
1	31	**MbRateCtrlFlag- RateControlCounterEnable (Encoder-only)** To enable the accumulation of bit allocation for rate control This field enables hardware Rate Control logic. The rest of the RC control fields are only valid when this field is set to 1. Otherwise, hardware ignores these fields. Note: To reset MB level rate control (QRC), we need to set both bits MbRateCtrlFlag and MbRateCtrlReset to 1 in the new slice **Value** — **Name** 0h — Disable 1h — Enable

MFC_MPEG2_SLICEGROUP_STATE

30 **MbRateCtrlReset- ResetRateControlCounter (Encoder-only)**

To reset the bit allocation accumulation counter to 0 to restart the rate control.

Value	Name	Description
0h	Disable	Not reset
1h	Enable	reset

29:28 **MbRateCtrlMode- RC Triggle Mode (Encoder-only)**

Value	Name	Description
00b		Always Rate Control, whereas RC becomes activeif sum_act > sum_target or sum_act < sum_target
01b		Gentle Rate Control, whereas RC becomes activeif sum_act > upper_midpt or sum_act < lower_midpt
10b		Loose Rate Control, whereas RC becomes activeif sum_act > sum_max or sum_act < sum_min
11b		Reserved

27:24 **MbRateCtrlParam- RC Stable Tolerance (Encoder-only)**

Format:	U4

This field specifies the tolerance required to deactivate RC once it has been triggered.

Value	Name
[0, 15]	

23 **RateCtrlPanicFlag - RC Panic Enable (Encoder-only)**

If this field is set to 1, RC enters panic modewhen sum_act > sum_max. RC Panic Type field controls what type of panic behavior is invoked.

Value	Name
0	Disable
1	Enable

22 **RateCtrlPanicType - RC Panic Type (Encoder-only)**

This field selects between two RC Panic methods. If it is set to 0, in panic mode, the macroblock QP is maxed out, setting to requested QP + QP_max_pos_mod. If it is set to 1, for an intra macroblock, AC CBPs are set to zero (note that DC CBPs are not modified). For inter macroblocks, AC and DC CBPs are forced to zero.

Value	Name	Description
0h		QP Panic
1h		CBP Panic

21 **Reserved**

Access:	RO
Format:	MBZ

20 **SkipConvDisabled - MB Type Skip Conversion Disable (Encoder-only)**

This field is only valid for a P or B slice. It must be zero for other slice types. Rules are provided in Section 2.3.3.1.6

<table>
<tr><th colspan="3">MFC_MPEG2_SLICEGROUP_STATE</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Enable</td><td>Enable skip type conversion</td></tr>
<tr><td>1h</td><td>Disable</td><td>Disable skip type conversion</td></tr>
</table>
</td></tr>
<tr><td></td><td>19</td><td><b>IsLastSliceGrp</b>
IsLastSliceGrp = 1 if the current slice group is the last slice group of a picture; 0 otherwise. It is used by the zero filling in the Minimum Frame Size test.</td></tr>
<tr><td></td><td>18</td><td><b>BitstreamOutputFlag - Compressed BitStream Output Disable Flag (Encoder-only)</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Enable</td><td>enable the writing of the output compressed bitstream</td></tr>
<tr><td>1h</td><td>Disable</td><td>disable the writing of the output compressed bitstream</td></tr>
</table>
</td></tr>
<tr><td></td><td>17</td><td><b>HeaderPresentFlag - Header Insertion Present in Bitstream (Encoder-only)</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable</td><td>no header insertion into the output bitstream buffer, in front of the current slice encoded bits</td></tr>
<tr><td>1h</td><td>Enable</td><td>header insertion into the output bitstream buffer is present, and is in front of the current slice encoded bits.</td></tr>
</table>
</td></tr>
<tr><td></td><td>16</td><td><b>SliceData PresentFlag - SliceData Insertion Present in Bitstream (Encoder-only)</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable</td><td>no Slice Data insertion into the output bitstream buffer</td></tr>
<tr><td>1h</td><td>Enable</td><td>Slice Data insertion into the output bitstream buffer is present.</td></tr>
</table>
</td></tr>
<tr><td></td><td>15</td><td><b>TailPresentFlag - Tail Insertion Present in bitstream (Encoder-only)</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td></td><td>no tail insertion into the output bitstream buffer, after the current slice encoded bits</td></tr>
<tr><td>1h</td><td></td><td>tail insertion into the output bitstream buffer is present, and is after the current slice encoded bits.</td></tr>
</table>
</td></tr>
<tr><td></td><td>14</td><td><b>FirstSliceHdrDisabled</b>
when this is on, the first slice header of the slice group is expected to be provided by the user via insertion command. PAK HW will skip it.</td></tr>
<tr><td></td><td>13</td><td><b>IntraSlice</b>
intra slice value included in slice headers, when IntraSliceFlag = 1.</td></tr>
<tr><td></td><td>12</td><td><b>IntraSliceFlag</b>
intra slice flag included in slice headers</td></tr>
<tr><td></td><td>11:8</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>7:4</td><td><b>SliceID[3:0] (Encoder-only)</b>
To identify the output data (coding information record) returned for rate control from PAK to ENC and VPP</td></tr>
</table>

MFC_MPEG2_SLICEGROUP_STATE

	3:2	**Reserved** Access: RO Format: MBZ
	1:0	**StreamID[1:0] (Encoder-only)** To identify the output data (coding information record) returned for rate control from PAK to ENC and VPP
2	31:24	**NextSgMbYcnt - also NextStartVertPos** Vertical count of the first MB in the next slice group (Encoder-only)Note: This field restricts total number of MB in the Y direction to 255 or less.
	23:16	**NextSgMbXcnt - also NextStartHorzPos**
	15:8	**FirstMbYcnt - also CurrStartVertPos** Format: U8 also CurrStartVertPos, Vertical count of the first MB in the current slice group (Encoder-only)
	7:0	**FirstMbXcnt - also CurrStartHorzPos** Format: U8 Horizontal count of the first MB in the current slice group (Encoder-only)
3	31:9	**Reserved** Access: RO Format: MBZ
	8	**SliceGroupSkip** Exists If: //Encoder Only Format: U1 All macroblocks are skipped
	7:6	**Reserved** Access: RO Format: MBZ
	5:0	**SliceGroupQp** Exists If: //Encoder Only Format: U6 Initial slice quality parameter
4	31:29	**Reserved** Access: RO Format: MBZ
	28:0	**BitstreamOffset - Indirect PAK-BSE Data Start Address (Write)** Exists If: //Encoder Only This field specifies the memory starting address (offset) to write out the compressed bitstream data from the BSE processing. This pointer is relative to the MFC Indirect PAK-BSE Object Base Address. It is a byte-aligned address for the AVC bitstream data in both CABAC/CAVLC Modes. For Write, there is no need to have a data length field. It is assumed the global memory bound

<table>
<tr><th colspan="3">MFC_MPEG2_SLICEGROUP_STATE</th></tr>
<tr><td></td><td></td><td>check specified in the IND_OBJ_BASE_ADDRESS command (Indirect PAK-BSE Object Access Upper Bound) will take care of any illegal write access. This field is only valid for AVC encode mode.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,512MB)</td><td></td></tr></table></td></tr>
<tr><td>5</td><td>31:24</td><td><b>MaxQpNegModifier - Magnitude of QP Max Negative Modifier (Encoder-only)</b><table><tr><td>Format:</td><td>U8</td></tr></table>This field specifies the lower limit of the QP modifier.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0, 51]</td><td></td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>MaxQpPosModifier - Magnitude of QP Max Positive Modifier (Encoder-only)</b><table><tr><td>Format:</td><td>U8</td></tr></table>This field specifies the upper limit of the QP modifier.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0, 51]</td><td></td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>ShrinkParam - Shrink Resistance (Encoder-only)</b><table><tr><td>Format:</td><td>U4</td></tr></table>This field specifies the additional points added each time decreased correction is invoked.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0, 15]</td><td></td></tr></table></td></tr>
<tr><td></td><td>11:8</td><td><b>Shrinkaram - Shrink Init (Encoder-only)</b><table><tr><td>Format:</td><td>U4</td></tr></table>This field specifies the initial points required to trip decreased control.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0, 15]</td><td></td></tr></table></td></tr>
<tr><td></td><td>7:4</td><td><b>GrowParam - Grow Resistance (Encoder-only)</b><table><tr><td>Format:</td><td>U4</td></tr></table>This field specifies the additional points added each time increased correction is invoked.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0, 15]</td><td></td></tr></table></td></tr>
<tr><td></td><td>3:0</td><td><b>GrowParam - Grow Init (Encoder-only)</b><table><tr><td>Format:</td><td>U4</td></tr></table>This field specifies the initial points required to trip increased control.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0, 15]</td><td></td></tr></table></td></tr>
<tr><td>6</td><td>31:24</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

MFC_MPEG2_SLICEGROUP_STATE

<table>
<tr><td></td><td>23:20</td><td><b>CorrectPoints - Correct 6 (Encoder-only)</b>
<table><tr><td>Format:</td><td>U4</td></tr></table>
This field specifies the points used in the lower most RC region when sum_act <= sum_min.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0, 15]</td><td></td></tr></table></td></tr>
<tr><td></td><td>19:16</td><td><b>CorrectPoints - Correct 5 (Encoder-only)</b>
<table><tr><td>Format:</td><td>U4</td></tr></table>
This field specifies the points used in the fifth RC region when sum_act > sum_min but <= lower_midpt.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0, 15]</td><td></td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>CorrectPoints - Correct 4 (Encoder-only)</b>
<table><tr><td>Format:</td><td>U4</td></tr></table>
This field specifies the points used in the fourth RC region when sum_act > lower_midpt but <= sum_target.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0, 15]</td><td></td></tr></table></td></tr>
<tr><td></td><td>11:8</td><td><b>CorrectPoints - Correct 3 (Encoder-only)</b>
<table><tr><td>Format:</td><td>U4</td></tr></table>
This field specifies the points used in the third RC region when sum_act > sum_target but <= upper_midpt.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0, 15]</td><td></td></tr></table></td></tr>
<tr><td></td><td>7:4</td><td><b>CorrectPoints - Correct 2 (Encoder-only)</b>
<table><tr><td>Format:</td><td>U4</td></tr></table>
This field specifies the points used in the second RC region when sum_act > upper_midpt but <= sum_max.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0, 15]</td><td></td></tr></table></td></tr>
<tr><td></td><td>3:0</td><td><b>CorrectPoints - Correct 1 (Encoder-only)</b>
<table><tr><td>Format:</td><td>U4</td></tr></table>
This field specifies the points used in the top most RC region when sum_act > sum_max
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0, 15]</td><td></td></tr></table></td></tr>
<tr><td>7</td><td>31:28</td><td><b>CV7 - Clamp Value 7 (Encoder-only)</b>
<table><tr><td>Exists If:</td><td>//Encoder Only</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>CV6 - Clamp Value 6 (Encoder-only)</b>
<table><tr><td>Exists If:</td><td>//Encoder Only</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFC_MPEG2_SLICEGROUP_STATE</th></tr>
<tr><td></td><td></td><td><table><tr><td>Format:</td><td>U4</td></tr></table></td></tr>
<tr><td></td><td>23:20</td><td><b>CV5 - Clamp Value 5 (Encoder-only)</b><table><tr><td>Exists If:</td><td>//Encoder Only</td></tr><tr><td>Format:</td><td>U4</td></tr></table></td></tr>
<tr><td></td><td>19:16</td><td><b>CV4 - Clamp Value 4 (Encoder-only)</b><table><tr><td>Exists If:</td><td>//Encoder Only</td></tr><tr><td>Format:</td><td>U4</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>CV3 - Clamp Value 3 (Encoder-only)</b><table><tr><td>Exists If:</td><td>//Encoder Only</td></tr><tr><td>Format:</td><td>U4</td></tr></table></td></tr>
<tr><td></td><td>11:8</td><td><b>CV2 - Clamp Value 2 (Encoder-only)</b><table><tr><td>Exists If:</td><td>//Encoder Only</td></tr><tr><td>Format:</td><td>U4</td></tr></table></td></tr>
<tr><td></td><td>7:4</td><td><b>CV1 - Clamp Value 1 (Encoder-only)</b><table><tr><td>Exists If:</td><td>//Encoder Only</td></tr><tr><td>Format:</td><td>U4</td></tr></table></td></tr>
<tr><td></td><td>3:0</td><td><b>CV0 - Clamp Value 0 (Encoder-only)</b>
If the magnitude of coefficients at locations assigned with CV0 (mapping shown below) exceeds 2CV0-1, they are replaced with2CV0-1. For coefficients at locations marked as 'none', no clamping is performed. The following mappings are only applied to luma and chromablocks\subblocks containing AC coefficiencts (blocks\sublocks with only DCcoeffs will not be clamped).
For 8x8 frame block, each coefficient is mapped to one of the eight CV values as following:<table>
<tr><td>none</td><td>none</td><td>CV7</td><td>CV6</td><td>CV5</td><td>CV4</td><td>CV3</td><td>CV3</td></tr>
<tr><td>none</td><td>CV7</td><td>CV6</td><td>CV5</td><td>CV4</td><td>CV3</td><td>CV3</td><td>CV2</td></tr>
<tr><td>CV7</td><td>CV6</td><td>CV5</td><td>CV4</td><td>CV3</td><td>CV3</td><td>CV2</td><td>CV2</td></tr>
<tr><td>CV6</td><td>CV5</td><td>CV4</td><td>CV3</td><td>CV3</td><td>CV2</td><td>CV2</td><td>CV1</td></tr>
<tr><td>CV5</td><td>CV4</td><td>CV3</td><td>CV3</td><td>CV2</td><td>CV2</td><td>CV1</td><td>CV1</td></tr>
<tr><td>CV4</td><td>CV3</td><td>CV3</td><td>CV2</td><td>CV2</td><td>CV1</td><td>CV1</td><td>CV0</td></tr>
<tr><td>CV3</td><td>CV3</td><td>CV2</td><td>CV2</td><td>CV1</td><td>CV1</td><td>CV0</td><td>CV0</td></tr>
<tr><td>CV3</td><td>CV2</td><td>CV2</td><td>CV1</td><td>CV1</td><td>CV0</td><td>CV0</td><td>CV0</td></tr>
</table>For 8x8 field block, each coefficient is mapped to one of the eight CV values as following:<table>
<tr><td>none</td><td>none</td><td>CV6</td><td>CV5</td><td>CV4</td><td>CV3</td><td>CV2</td><td>CV1</td></tr>
<tr><td>none</td><td>CV7</td><td>CV6</td><td>CV5</td><td>CV4</td><td>CV3</td><td>CV2</td><td>CV1</td></tr>
<tr><td>CV7</td><td>CV6</td><td>CV5</td><td>CV4</td><td>CV3</td><td>CV3</td><td>CV2</td><td>CV1</td></tr>
<tr><td>CV7</td><td>CV6</td><td>CV5</td><td>CV4</td><td>CV3</td><td>CV2</td><td>CV2</td><td>CV1</td></tr>
<tr><td>CV6</td><td>CV5</td><td>CV4</td><td>CV4</td><td>CV3</td><td>CV2</td><td>CV1</td><td>CV0</td></tr>
</table></td></tr>
</table>

MFC_MPEG2_SLICEGROUP_STATE

CV6	CV5	CV4	CV3	CV2	CV2	CV1	CV0
CV5	CV5	CV4	CV3	CV2	CV1	CV1	CV0
CV5	CV5	CV4	CV3	CV2	CV1	CV1	CV0

MFD_AVC_BSD_OBJECT

MFD_AVC_BSD_OBJECT

Source:	VideoCS
Length Bias:	2

The MFD_AVC_BSD_OBJECT command is the only primitive command for the AVC Decoding Pipeline. The same command is used for both CABAC and CAVLD modes. The Slice Data portion of the bitstream is loaded as indirect data object. Before issuing a MFD_AVC_BSD_OBJECT command, all AVC states of the MFD Engine need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of a MFD_AVC_BSD_OBJECT command.

Context switch interrupt is not supported by this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_AVC_BSD_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 5h Excludes DWord (0,1) = 0005 Format: =n
1	31:0	**Indirect BSD Data Length** Format: U32 This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address. AVC Short Format: It is the length in bytes of the bitstream data for the current slice, including Slice Header + Slice Data

MFD_AVC_BSD_OBJECT		
		+ Emulation Prevention Bytes + any filling trailing zeros after the last MB. Hardware ignores the contents after the last non-zero byte. Trailing zero is allowed and handled correctly in both CABAC and CAVLC modes.
2	31:29	**Reserved** Access: RO Format: MBZ
	28:0	**Indirect BSD Data Start Address** Format: U29 This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the **MFD Indirect Object Base Address**. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the AVC bitstream data in both CABAC/CAVLD Modes. In implementing a phantom slice at the end of a picture for automatic error concealment, this field should set to 0. It includes the NAL Header (the NAL Header does not need to perform EMU detection). For AVC Base Layer, it is a single byte. But for MVC, the NAL Header is 4 Bytes long. These NAL Header Unit must be passed to HW in the compressed bitstream buffer. **Value** | **Name** [0,512MB) |
3..5	95:0	**Inline Data** Format: **Inline Data Description for MFD_AVC_BSD_Object** All the required Slice Header parameters and error handling settings are captured as InLine Data of the AVC_BSD_OBJECT command. It has a fixed size of 3 DWs. Its definition is described in the following section: Inline Data Description.
6	31:0	**Reserved** Access: RO Format: MBZ

MFD_AVC_DPB_STATE

MFD_AVC_DPB_STATE

Source:	VideoCS
Length Bias:	2

This is a frame level state command used only in AVC Short Slice Bitstream Format VLD mode.RefFrameList[16] of interface is replaced with intel Reference Picture Addresses[16] of MFX_PIPE_BUF_ADDR_STATE command. The LongTerm Picture flag indicator of all reference pictures are collected into LongTermPic_Flag[16].FieldOrderCntList[16][2] and CurrFieldOrderCnt[2] of interface are replaced with intel POCList[34] of MFX_AVC_DIRECTMODE_STATE command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 6h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 9h Excludes DWord (0,1) Format: =n
1	31:16	**LongTermFrame_Flag[16][1 bit]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 1 bit per reference frame. **Value** **Name** 1 the picture is a long term reference picture 0 the picture is a short term reference picture

<table>
<tr><th colspan="3">MFD_AVC_DPB_STATE</th></tr>
<tr><td></td><td>15:0</td><td><b>Non-ExistingFrame_Flag[16][1 bit]</b>
One-to-one correspondence with the entries of the Intel RefFrameList[16]. 1 bit per reference frame.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td>INVALID</td><td>the reference picture in that entry of RefFrameList[] does not exist anymore.</td></tr>
<tr><td>0</td><td>VALID</td><td>the reference picture in that entry of RefFrameList[] is a valid reference</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>When an element of the list of frames is not relevant (e.g., due to the corresponding reference entry being empty or being marked as "not used for reference"), the value of the corresponding bit of NonExistingFrameFlags shall be set to 0.</td></tr>
</table></td></tr>
<tr><td>2</td><td>31:0</td><td><b>UsedForReference_Flag[16][2 bits]</b>
One-to-one correspondence with the entries of the Intel RefFrameList[16]. 2 bits per reference frame.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NOT_REFERENCE</td><td>indicates a frame is "not used for reference".</td></tr>
<tr><td>1</td><td>TOP_FIELD</td><td>bit[0] indicates that the top field of a frame is marked as "used for reference".</td></tr>
<tr><td>2</td><td>BOTTOM_FIELD</td><td>bit[1] indicates that the bottom field of a frame is marked as "used for reference".</td></tr>
<tr><td>3</td><td>FRAME</td><td>bit[1:0] indicates that a frame (or field pair) is marked as "used for reference".</td></tr>
</table></td></tr>
<tr><td>3..10</td><td>255:0</td><td><b>LTSTFrameNumList[16][16 bits]</b>
One-to-one correspondence with the entries of the Intel RefFrameList[16]. 16 bits per reference frame. Depending on the corresponding LongTermFrame_Flag[], the content of this field is interpreted differently.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td>LongTermFrame_Flag[i]</td><td>LTSTFrameNumList[i] represent LongTermFrameIdx.</td></tr>
<tr><td>0</td><td>ShortTermFrame_Flag[i]</td><td>LTSTFrameNumList[i]represent Short Term Picture FrameNum.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>When an element of the list of frames is not relevant (e.g., due to the corresponding reference entry being empty or being marked as "not used for reference"), the value of the LTSTFrameNumList entry shall be set to 0.</td></tr>
</table></td></tr>
<tr><td>11..18</td><td>255:0</td><td><b>ViewIDList[16][16 bits]</b>
One-to-one correspondence with the entries of the Intel RefFrameList[16]. 16 bits per reference frame. The view ids are 10-bits, the upper 6 bits are ignored."000000" & ViewId1[9:0] & "000000" & ViewId0[9:0]
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>When an Intel RefFrameList[i] is not a valid entries, Viewid should be set to 0x00</td></tr>
</table></td></tr>
</table>

MFD_AVC_DPB_STATE		
19..22	127:0	**ViewOrderListL0[16][8 bits]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 8 bits per reference frame. The view order need 4-bits, the upper 4 bits are ignored.0000 & ViewOrder3[3:0] & 0000 & ViewOrder2[3:0] & 0000 & ViewOrder1[3:0] & 0000 & ViewOrder0[3:0] **Programming Notes** When the ViewOrderListL0[i] is not an valid inter-view reference, its corresponding ViewOrder should be set to 0xF Since only interview with the same polarity will be used, there is no need to have field bit in this list. Hardware is going to append correct polarity bit as needed.
23..26	127:0	**ViewOrderListL1[16][8 bits]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 8 bits per reference frame. The view order need 4-bits, the upper 4 bits are ignored.0000 & ViewOrder3[3:0] & 0000 & ViewOrder2[3:0] & 0000 & ViewOrder1[3:0] & 0000 & ViewOrder0[3:0] **Programming Notes** When the ViewOrderListL1[i] is not a valid inter-view reference, its corresponding ViewOrder should be set to 0xF Since only interview with the same polarity will be used, there is no need to have field bit in this list. Hardware is going to append correct polarity bit as needed.

MFD_AVC_PICID_STATE

<table>
<tr><th colspan="3">MFD_AVC_PICID_STATE</th></tr>
<tr><td colspan="3">Source: VideoCS
Length Bias: 2</td></tr>
<tr><td colspan="3">This is a frame level state command used for both AVC Long and Short Format in VLD mode. PictureID[16] contains the pictureID of each reference picture (16 maximum) so hardware can uniquely identify the reference picture across frames (this will be used for DMV operation).This command will be needed for both short and long format.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline</b><table><tr><td>Default Value:</td><td>2h MFX_MULTI_DW</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><b>Media Command Opcode</b><table><tr><td>Default Value:</td><td>1h MFD_AVC_DPB_STATE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>23:21</td><td><b>SubOpcode A</b><table><tr><td>Default Value:</td><td>1h DEC</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>20:16</td><td><b>SubOpcode B</b><table><tr><td>Default Value:</td><td>5h MEDIA_</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b><table><tr><td>Default Value:</td><td>0008h Excludes DWord (0,1)</td></tr><tr><td>Format:</td><td>=n</td></tr></table></td></tr>
<tr><td>1</td><td>31:1</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>PictureID Remapping Disable</b><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>AVC decoder will use 16 bits Picture ID to handle DMV and identify the reference picture</td><td>Desc</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFD_AVC_PICID_STATE</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>1h</td><td>AVC decoder will use 4 bits FrameStoreID (index to RefFrameList) to handle DMV and identify the reference picture</td><td>Desc</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>If Picture ID Remapping Disable is "1", PictureIDList will not be used.</td></tr>
</table>
</td></tr>
<tr><td>2..9</td><td>255:0</td><td><b>PictureIDList[16][16 bits]</b>
 One-to-one correspondence with the entries of the Intel RefFrameList[16]. 16 bits per reference frame. PictureID of each Frame uniquely identifies the reference picture across frames. The same number cannot be reused until the reference picture is completely retired(no longer used for reference)When an element of the list of frames is not relevant (e.g., due to the corresponding reference entry being empty or being marked as "not used for reference"), the value of the LTSTFrameNumList entry shall be set to 0.</td></tr>
</table>

MFD_AVC_SLICEADDR

MFD_AVC_SLICEADDR

Source:	VideoCS
Length Bias:	2

This is a Slice level command used only for AVC Short Slice Bitstream Format VLD mode. When decoding a slice, H/W needs to know the last MB of the slice has reached in order to start decoding the next slice. It also needs to know if a slice is terminated but the last MB has not reached, error concealment should be invoked to generate those missing MBs. For AVC Short Format, the only way to know the last MB position of the current slice, H/W needs to snoop into the next slice's start MB address (a linear address encoded in the Slice Header). Since each BSD Object command can have only one indirect bitstream buffer address, this command is added to help H/W to snoop into the next slice's slice header and retrieve its Start MB Address. This command will take the next slice's bitstream buffer address as input (exactly the same way as a BSD Object command), and parse only the first_mb_in_slice syntax element. The result will stored inside the H/W, and will be used to decode the current slice specified in the BSD Object command. Only the very first few bytes (max 5 bytes for a max 4K picture) of the Slice Header will be decoded, the rest of the bitstream are don't care. This is because the first_mb_in_slice is encoded in Exponential Golomb, and will take 33 bits to represent the max 256 x 256 = 64K-1 value. The indirect data of MFD_AVC_SLICEADDR is a valid BSD object and is decoded as in BSD OBJECT command. The next Slice Start MB Address is also exposed to the MMIO interface. The Slice Start MB Address (first_mb_in_slice) is a linear MB address count; but it is translated into the corresponding 2D MB X and Y raster position, and are stored internally as NextSliceMbY and NextSliceMbX.

DWord	Bit	Description		
0	31:29	**Command Type**		
			Default Value:	3h PARALLEL_VIDEO_PIPE
			Format:	OpCode
	28:27	**Pipeline**		
			Default Value:	2h MFD_AVC_ SLICEADDR
			Format:	OpCode
	26:24	**Media Command Opcode**		
			Default Value:	1h AVC_DEC
			Format:	OpCode
	23:21	**SubOpcode A**		
			Default Value:	1h
			Format:	OpCode
	20:16	**SubOpcode B**		
			Default Value:	7h
			Format:	OpCode
	15:12	**Reserved**		
			Access:	RO
			Format:	MBZ

<table>
<tr><th colspan="3">MFD_AVC_SLICEADDR</th></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b><table><tr><td>Default Value:</td><td>2h</td></tr><tr><td>Format:</td><td>=n</td></tr></table></td></tr>
<tr><td>1</td><td>31:0</td><td><b>Indirect BSD Data Length</b><table><tr><td>Format:</td><td>U32</td></tr></table>This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. Driver always programs this up to 5 bytes; for bitstream less than 5 bytes, driver program the lesser value. (Emulation Prevention Byte should never happen for the first 5 bytes when the max picture size can only be 4Kx4K)It is the length in bytes of the bitstream data for the current slice, including Slice Header + Slice Data + Emulation Prevention Bytes + any filling trailing zeros after the last MB. Hardware ignores the contents after the last non-zero byte. Trailing zero is allowed and handled correctly in both CABAC and CAVLC modes.</td></tr>
<tr><td rowspan="2">2</td><td>31:29</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>28:0</td><td><b>Indirect BSD Data Start Address</b>
This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the AVC bitstream data in both CABAC/CAVLD Modes. In implementing a phantom slice at the end of a picture for automatic error concealment, this field should set to 0.It includes the NAL Header Byte. (but does not perform EMU detection). Must provide a valid MB address, even if error. MB must be clamped to within a pic boundary.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,512MB)</td><td></td></tr></table></td></tr>
<tr><td rowspan="3">3</td><td>31:13</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>12:9</td><td><b>Reserved</b></td></tr>
<tr><td>8</td><td><b>AVC NAL Type First Byte Override Bit</b><table><tr><td>Format:</td><td>U1</td></tr></table>This bit indicates hardware should use the NAL Type (provided below) programmed by driver instead of using the one from bitstream. The NAL byte from bitstream will not be correct.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Use Bitstream Decoded NAL Type</td><td>NAL Type should come from first byte of decoded bitstream.</td></tr><tr><td>1</td><td>Use Driver Programmed NAL Type</td><td>NAL Type should come from "Driver Provided NAL Type Values" programmed by driver.</td></tr></table></td></tr>
</table>

MFD_AVC_SLICEADDR		
	7:0	**Driver Provided NAL Type Value** Format: U8 This will replace the first byte of the NAL unit, containing forbidden_zero_bit, nal_ref_idc, and nal_unit_type. **Programming Notes** This byte should be ignored if AVC NAL Type First Byte Override Bit is programmed to 0

MFD_IT_OBJECT

MFD_IT_OBJECT		
Source:	VideoCS	
Length Bias:	2	
All weight mode (default and implicit) are mapped to explicit mode. But the weights come in either as explicit or implicit.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_IT_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Format: =n

Value	Name	Description	Exists If
0Ch	AVC	There are total of 7 inline DWs for AVC (in addition to DW0-DW6 here)	//mode=='AVC'
10h	VC1	There are total of 11 inline DWs for VC1 (in addition to DW0-DW6 here)	//mode=='VC1'
0Bh	MPEG2	There are total of 6 inline DWs for AVC (in addition to DW0-DW6 here)	//mode=='MPEG2'

<table>
<tr><th colspan="3">MFD_IT_OBJECT</th></tr>
<tr><td>1</td><td>31:10</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>9:0</td><td><b>Indirect IT-MV Data Length</b><table><tr><td>Format:</td><td>U10</td></tr></table>This field provides the length in bytes of the indirect data, which contains all the MVs for the current MB (in any partitioning and subpartitioning form). A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect IT-MV Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address. AVC-IT Mode: It must be DWord aligned (since each MV is 4bytes in size)Driver has to derived this field from MVsize (MVquantity in DXVA, exact size) *4 bytes per MV. This field is only valid in AVC decoder IT mode (VC1 and MPEG uses inline MV data).</td></tr>
<tr><td>2</td><td>31:29</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>28:0</td><td><b>Indirect IT-MV Data Start Address Offset</b>
This field specifies the memory starting address (offset) of the MV data to be fetched into the IT pipeline for processing. This pointer is relative to the Indirect IT-MV Object Base Address. Hardware ignores this field if indirect data is not present, i.e. the Indirect MV Data Length is set to 0. Alignment of this address depends on the mode of operation. AVC-IT Mode: It must be DWord aligned (since each MV is 4 bytes in size). This field is only valid in AVC decoder IT mode (VC1 and MPEG uses inline MV data).<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,512MB)</td><td></td></tr></table></td></tr>
<tr><td>3</td><td>31:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>Indirect IT-COEFF Data Length</b>
This field provides the length in bytes of the indirect data, which contains all the non-zero coefficients for the current MB. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect IT-COEFF Data Start Address field is ignored. Since each IT-COEFF data is 1 DW in size, with 12 bits, this field can be extended to support up to 4:4:4 format.(256 pixel * 3 byte pixel components * 4 bytes per coeff).This field must be integer multiple of 16-bytes for AVC (since each coefficient is 4 bytes in size).This field is only valid in AVC, VC1, MPEG2 decoder IT mode.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,3072]</td><td>In bytes [0, 256*3*4]</td></tr></table></td></tr>
<tr><td>4</td><td>31:29</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFD_IT_OBJECT</th></tr>
<tr><td></td><td>28:0</td><td><b>Indirect IT-COEFF Data Start Address Offset</b>
This field specifies the memory starting address (offset) of the coeff data to be loaded into the IT pipeline for processing. This pointer is relative to the Indirect IT-COEFF Object Base Address. Hardware ignores this field if indirect IT-COEFF data is not present, i.e. the Indirect IT-COEFF Data Length is set to 0.This field must be DW aligned, since each coefficient is 4 bytes in size. Driver will determine the Num of EOB 4x4/8x8 must match the block cbp flags, if not match, hardware cannot hang - add error handling. This field is only valid in AVC, VC1, MPEG2 decoder IT mode.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,512MB)</td><td></td></tr></table></td></tr>
<tr><td>5</td><td>31:6</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Indirect IT-DBLK Control Data Length</b>
<table><tr><td>Format:</td><td>U6</td></tr></table>
This field provides the length in bytes of the indirect data, which contains all the deblocker control information for the current MB (in 4x4 sub-block partitioning). A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect IT-DBLK Data Start Address field is ignored. This field must have the same alignment as the Indirect IT-DBLK Data Start Address. It must be DWord aligned. Each Deblock Control Data record is 48 bytes or 12 DWords in size. This field is only valid in AVC decoder IT mode.</td></tr>
<tr><td>6</td><td>31:29</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>28:0</td><td><b>Indirect IT-DBLK Control Data Start Address Offset</b>
<table><tr><td>Format:</td><td>IndirectObjectOffset[28:0]</td></tr></table>
This field specifies the memory starting address (offset) of the Deblocker control data to be fetched into the IT Pipeline for processing. This pointer is relative to the Indirect IT-DBLK Object Base Address. Hardware ignores this field if indirect data is not present, ie. The indirect IT-DBLK Control Data Length is set to 0.It must be DWord aligned. Each Deblock Control Data record is 48 bytes or 12 DWords in size. This field is only valid in AVC decoder IT mode.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,512MB)</td><td></td></tr></table></td></tr>
<tr><td>7..12
<b>Exists if:</b> //mode == 'MPEG2'
<b>Programming Notes:</b>
MPEG2--There are 6 addition</td><td>191:0</td><td><b>MPEG2 Inline Data</b>
Union for all 3 codecs. Includes IT, MC, IntraPred inline data as well as Deblocker control information.
AVC-IT Modes: Hardware interprets this data in the specified format.
VC1-IT Modes: Hardware interprets this data in the specified format. MV inline.
MPEG2-IT Modes: Hardware interprets this data in the specified format. (IS mode) MV inline For AVC there 7 DWords of inline data, hence N is equal to 13.</td></tr>
</table>

MFD_IT_OBJECT		
DWs so n = 12		
7..13 **Exists if:** //mode == 'AVC' **Programming Notes:** AVC--There are 7 addition DWs so n = 13	223:0	**AVC Inline Data** Union for all 3 codecs. Includes IT, MC, IntraPred inline data as well as Deblocker control information. AVC-IT Modes: Hardware interprets this data in the specified format. VC1-IT Modes: Hardware interprets this data in the specified format. MV inline MPEG2-IT Modes: Hardware interprets this data in the specified format. (IS mode) MV inline For AVC there 7 DWords of inline data, hence N is equal to 13.
7..17 **Exists if:** //mode == 'VC1' **Programming Notes:** VC1--There are 11 addition DWs so n = 17	351:0	**VC1 Inline Data** Union for all 3 codecs. Includes IT, MC, IntraPred inline data as well as Deblocker control information. AVC-IT Modes: Hardware interprets this data in the specified format. VC1-IT Modes: Hardware interprets this data in the specified format. MV inline. MPEG2-IT Modes: Hardware interprets this data in the specified format. (IS mode) MV inline For AVC there 7 DWords of inline data, hence N is equal to 13.

MFD_JPEG_BSD_OBJECT

MFD_JPEG_BSD_OBJECT		
Source:	VideoCS	
Length Bias:	2	
Exists If:	//Decoder	

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_JPEG_BSD_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 7h JPEG_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 004h Excludes DWord (0,1) Format: =n
1	31:0	**Indirect Data Length** . It is the length in bytes of the bitstream data for the current Scan. It includes the first byte of the first MCU and the last non-zero byte of the last MCU in the Scan. Specifically, the zero-padding bytes (if present) are excluded. Hardware ignores the contents after the last non-zero byte.
2	31:29	**Reserved** Access: RO Format: MBZ
	28:0	**Indirect Data Start Address** Format: IndirectObjectOffset[28:0]

<table>
<tr><th colspan="3">MFD_JPEG_BSD_OBJECT</th></tr>
<tr><td></td><td></td><td>This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the BSD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the JPEG bitstream data</td></tr>
<tr><td>3</td><td>31:29</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>28:16</td><td><b>Scan Horizontal Position</b><table><tr><td>Format:</td><td>U13</td></tr></table>This field indicates the horizontal position (in block units) of the first MCU in the Scan.</td></tr>
<tr><td></td><td>15:13</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>12:0</td><td><b>Scan Vertical Position</b><table><tr><td>Format:</td><td>U13</td></tr></table>This field indicates the vertical position (in block units) of the first MCU in the Scan.</td></tr>
<tr><td>4</td><td>31</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Interleaved</b><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Non-Interleaved</td><td>one component in the Scan</td></tr><tr><td>1</td><td>Interleaved</td><td>multiple components in the Scan</td></tr></table></td></tr>
<tr><td></td><td>29:27</td><td><b>Scan Components</b>
Bit0: YBit1: UBit2: V For example, if non-interleaved Y, then it will be set to 001b. If interleaved Y, U, and V, it will be set to 111b.</td></tr>
<tr><td></td><td>26</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>25:0</td><td><b>MCU Count</b><table><tr><td>Format:</td><td>U26</td></tr></table>This field indicates the number of MCUs in the Scan.</td></tr>
<tr><td>5</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>RestartInterval(16 bit)</b><table><tr><td>Format:</td><td>U16</td></tr></table>Specifies the number of MCU in restart interval. Valid values are 1->0xFFFFValue of 0 implies that all the SCAN have only one ECS.</td></tr>
</table>

MFD_MPEG2_BSD_OBJECT

MFD_MPEG2_BSD_OBJECT

Source:	VideoCS
Length Bias:	2

Different from AVC and VC1, MFD_MPEG2_BSD_OBJECT command is pipelinable. This is for performance purpose as in MPEG2 a slice is defined as a group of MBs of any size that must be within a macroblock row. Slice header parameters are passed in as inline data and the bitstream data for the slice is passed in as indirect data. Of the inline data, slice_horizontal_position and slice_vertical_position determines the location within the destination picture of the first macroblock in the slice. The content in this command is identical to that in the MEDIA_OBJECT command in VLD mode described in the Media Chapter.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_MPEG2_BSD_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 3h MPEG2_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 0003h Excludes DWord (0,1) Format: =n
1	31:0	**Indirect BSD Data Length** Format: U32 It is the length in bytes of the bitstream data for the current slice. It includes the first byte of the first macroblock and the last non-zero byte of the last macroblock in the slice. Specifically, the zero-padding bytes (if present) and the next start-code are excluded. This field is sized to support beyond MPEG-2 MP@HL bitstream (<4K). According to Table 8-6 of ISO/IEC 13818-2,

MFD_MPEG2_BSD_OBJECT		
		the maximum number of bits per macroblock for 4:2:0 is 4608. So the maximum slice size for 4K x 4K is 4608 * 256 / 8 = 147,456 bytes (0x24000), which requires 18 bits. **Programming Notes** As MPEG-2 spec does not post any limitation of the size of zero-padding bytes, it is possible to have a slice data with large length (including zero-padding bytes). As the data beyond 0x10E00 would only be zero bytes for a valid slice data zero-padding restriction is removed
2	31:29	**Reserved** Access: RO Format: MBZ
	28:0	**Indirect Data Start Address** Format: IndirectObjectOffset[28:0] This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the BSD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the MPEG2 VLD bitstream data This address points to the first byte of the MB layer data, i.e. not including slice header.
3..4	63:0	**Inline Data** Format: **MFD_MPEG2_BSD_OBJECT Inline Data Description** All the required Slice Header parameters and error handling settings are captured as MPEG2_BSD_OBJECT Inline Data Descriptor structures. It has a fixed size of 2 DWs. Its definition is described in the next section.

MFD_VC1_BSD_OBJECT

MFD_VC1_BSD_OBJECT

Source:	VideoCS
Length Bias:	2

The MFD_VC1_BSD_OBJECT command is the only primitive command for the VC1 Decoding Pipeline. The macroblock data portion of the bitstream is loaded as indirect data object. Before issuing a MFD_VC1_BSD_OBJECT command, all VC1 states of the MFD Engine need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of a MFD_VC1_BSD_OBJECT command.VC1 deblock filter kernel cross the slice boundary if in the last MB row of a slice, so need to know the last MB row of a slice to disable the edge mask. There is why VC1 BSD hardware need to know the end of MB address for the current slice. As such no more phantom slice is needed for VC1, as long as the driver will program both start MB address in the current slice and the start MB address of the next slice. As a result, we can also support multiple picture state commands in between slices.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 0003h Excludes DWord (0,1) Format: =n
1	31:24	**Reserved** Access: RO Format: MBZ

MFD_VC1_BSD_OBJECT

<table>
<tr><td></td><td>23:0</td><td><b>Indirect BSD Data Length</b>
<table><tr><td>Format:</td><td>U24</td></tr></table>
This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address. Long Format : It is the length in bytes of the bitstream data for the current slice/picture. It includes the first byte of the first macroblock and the last byte of the last macroblock in the slice/picture. Specifically, the zero-padding bytes (if present) and the next start-code are excluded. Hardware ignores the contents after the last non-zero byte (trailing zeros). This field is sized to support VC1 AP@L4 Level bitstream. It includes the byte that contains the First MB Bit Offset Short Format : It is the length in bytes of the bitstream data for the current slice, including Picture/Slice Header + Emulation Prevention Bytes + any filling trailing zeros after the last MB. Hardware ignores the contents after the last non-zero byte. Trailing zero is allowed and handled correctly.</td></tr>
<tr><td>2</td><td>31:29</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>28:0</td><td><b>Indirect Data Start Address</b>
<table><tr><td>Format:</td><td>IndirectObjectOffset[28:0]</td></tr></table>
This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VC1 bitstream data.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,512MB)</td><td></td></tr></table></td></tr>
<tr><td>3</td><td>31:24</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>Slice Start Vertical Position</b>
This field specifies the position in y-direction of the first macroblock in the Slice in unit of macroblocks. For SecondField this value is reset to zero as opposed to the VC1 spec Ref: 9.1.2 Slice Layer. This field is for both Long and Short VC1 Interface Format.</td></tr>
<tr><td></td><td>15:9</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>8:0</td><td><b>Next Slice Vertical Position</b>
This field specifies the position in y-direction of the first macroblock in the next Slice in unit of macroblocks. This field is primarily used for error concealment. In the case that current slice is the last slice, this field should set to the height of picture (since y-direction is zero-based numbering) This field is maintained and provided by the driver for both Long and Short VC1 Interface Format.</td></tr>
</table>

<table>
<tr><th colspan="3">MFD_VC1_BSD_OBJECT</th></tr>
<tr><td rowspan="5">4</td><td>31:16</td><td><b>First_MB_Byte_Offset of Slice Data or Slice Header</b>
For DXVA2 VC1 Short Format only It gives the byte offset to locate the first MB data in the bitstream for a slice, relative to the Indirect BSD Data Start Address.</td></tr>
<tr><td>15:5</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>4</td><td><b>Emulation Prevention Byte Present</b><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td></td><td>H/W needs to perform Emulation Byte Removal</td></tr><tr><td>1h</td><td></td><td>H/W does not need to perform Emulation Byte Removal</td></tr></table></td></tr>
<tr><td>3</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>2:0</td><td><b>FirstMbBitOffset (First Macroblock Bit Offset)</b><table><tr><td>Format:</td><td>U3</td></tr></table>This field provides the bit offset of the first macroblock of the Slice in the first byte of the input compressed bitstream. It is used with First_MB_Byte_Offset for non-byte aligned position.</td></tr>
</table>

MFD_VC1_LONG_PIC_STATE

MFD_VC1_LONG_PIC_STATE

Source: VideoCS
Length Bias: 2

MFX_VC1_LONG PIC_STATE command encapsulates the decoding parameters that are read or derived from bitstream syntax elements above (inclusive) picture header layer. These parameters are static for a picture and when slice structure is present, these parameters are not changed from slice to slice of the same picture. Hence, this command is only issued at the beginning of processing a new picture and prior to the VC1_*_OBJECT command. The values set for these state variables are retained internally across slices. Only the parameters needed by hardware (BSD unit) to decode bit sequence for the macroblocks in a picture layer or a slice layer are presented in this command. Other parameters such as the ones used for inverse transform or motion compensation are provided in MFX_VC1_PRED_PIPE_STATE command. This Long interface format is intel proprietary interface. Driver will need to perform addition operations to generate all the fields in this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_VC1_LONG_PIC_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 1h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">MFD_VC1_LONG_PIC_STATE</th></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b><table><tr><td>Default Value:</td><td>0004h Excludes DWord (0,1)</td></tr><tr><td>Format:</td><td>=n</td></tr></table></td></tr>
<tr><td>1</td><td>31:24</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>PictureHeightInMBsMinus1 (Picture Height Minus 1 in Macroblocks)</b><table><tr><td>Format:</td><td>U8</td></tr></table>This field indicates the height of the picture in unit of macroblocks. For example, for a 1920x1080 frame picture, PictureHeightInMBs equals 68 (1080 divided by 16, and rounded up, i.e. effectively specified as 1088 instead). This field is used in VLD and IT modes.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>[0,255]</td><td>ValueHeight</td><td>Represents 1 MB to 256 MB</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>Note: Even though the Advanced Profile allows frame dimensions (width, height) to not be aligned to macroblock boudary, it doesn't affect the bitstream decoding. And it is preferable to use 'intermediate buffer' that is macroblock aligned for decoding. In order to simplify the out-of-bound reference pixel access, the out-of-bound extrapolation rule in VC1 spec can be used to expand the expected decoded frame to the intermediate buffer dimension.</td></tr></table></td></tr>
<tr><td></td><td>15:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>PictureWidthInMBsMinus1 (Picture Width Minus 1 in Macroblocks)</b><table><tr><td>Format:</td><td>U8-1</td></tr></table>This field indicates the width of the picture in unit of macroblocks. For example, for a 1920x1080 frame picture, PictureWidthInMBs equals 120 (1920 divided by 16). This field is used in VLD and IT modes<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>[0,255]</td><td>ValidWidth</td><td>Represents 1 MB to 256 MB</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFD_VC1_LONG_PIC_STATE</th></tr>
<tr><td>2</td><td>31:24</td><td><b>Bitplane Buffer Pitch Minus 1</b>
<table>
<tr><td>Format:</td><td>U8-1</td></tr>
</table>
Specifies the bitplane buffer pitch in (#Bytes - 1). Bitplane buffer is a linear buffer. It is needed only when the bitplane is not encoded as raw, and therefore is present in the header explicitly. In VC1 Long Format, it is written by an application and later read by the HW. But in VC1 Short Format, it is written and read by H/W only.This field is specified for better performance
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>[0h, FFh]</td><td></td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>The pitch must be equal to PictureWidthInMBs/2. For VC1 Long Format : The pitch must be equal to PictureWidthInMBs/2.For VC1 Short Format : If Pic Width is less than or equal to 2K pixels, bitplane pitch is set to 64 (one cacheline; programmed as 63) bytes per MB row. If Pic Width is greater than 2K pixels, bitplane pitch is set to 128 (two cachelines; programmed as 127) bytes per MB row. This field is not used in IT mode, used in VLD mode only.For VC1 Short Format, the bitplane specification is between H/W and Driver only. For Long Format, application is responsible for allocation with the driver.</td></tr>
</table></td></tr>
<tr><td></td><td>23:16</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>15</td><td><b>DmvSurfaceValid</b>
Indicated when the DMV read surface is valid. This surface stored the direct motion vectors and Mb type. This field is set for B pictures that can refer to a previous P picture for DMV. If there is an I-picture before a B (in decoding order) then this field is not set (as a result, zero's DMV's will be assumed while decoding the B picture. That is, there is no explicit DMV buffer for an I-picture).When the current picture being decoded is an I, P or BI, this bit is set to 0, since there is no DMV read in these picture decoding process. This field is not used in IT mode, used in VLD mode only.</td></tr>
<tr><td></td><td>14</td><td><b>ImplicitQuantizer</b>
Derived by driver from QUANTIZER. This field is used in intel VC1 VLD Long Format only, not used in IT and VC1. This bit is set to 1 when syntax element QUANTIZER=0, else its set to 0</td></tr>
<tr><td></td><td>13</td><td><b>Interpolation Rounder Contro</b>
Used only in MC operation. This field specifies the rounding control value used in interpolation operation of motion prediction process. This field is used in VLD and IT modes.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This bit field is taken from bR control in PictureParameters data structure</td></tr>
</table></td></tr>
</table>

MFD_VC1_LONG_PIC_STATE

12

SyncMarker

Indicates whether sync markers are enabled/disabled. If enable, sync markers "may be" present in the current video sequence being decoded. It is a sequence level syntax element and is valid only for Simple and Main Profiles.

Value	Name	Description
0h	Not Present	Sync Marker is not present in the bitstream
1h	Maybe present	Sync Marker maybe present in the bitstream

Programming Notes
This field is only valid in VLD mode. For Simple Profile, SyncMarker must set to 0.For Main Profile, SyncMarker can be set to 0 or 1.This field is used in both intel and MS VLD interface, but not used in IT mode.

11:8

Motion Vector Mode

This field indicates one of the following motion compensation interpolation modes for P and B pictures. The MC interpolation modes apply to prediction values of luminance blocks and are always in quarter-sample. For chrominance blocks, it always performs bilinear interpolation with either half-pel or quarter-pel precision. Before the polarity of Chroma Half-pel or Q-pel is reversed from Spec, now I have fixed it to match with VC1 Spec.

Value	Name	Description
0XX0b		Chroma Quarter -pel + Luma bicubic. (can only be 1MV)
0XX1b		Chroma Half-pel + Luma bicubic. (can be 1MV or 4MV)
1XX0b		Chroma Quarter -pel + Luma bilinear. (can only be 1MV)
1XX1b		Chroma Half-pel + Luma bilinear

Programming Notes
Bits 11:8 are taken from bMVprecisionAndChromaRelation in PictureParameters data structure.Bit 11 of Motion Vector Mode = 1 for Luma Bilinear MC; = 0 for Luma Bicubic MCBit 8 of Motion Vector Mode = 1 for half-sample Chroma motion = 0 for quarter-sample Chroma motion. This field is used in both VLD and IT modes.

<table>
<tr><th colspan="3">MFD_VC1_LONG_PIC_STATE</th></tr>
<tr><td></td><td>7</td><td><b>RangeReductionScale</b>

This field specifies whether the reference picture pixel values should be scaled up or scaled down on-the-fly, if RangeReduction is Enabled.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td></td><td>Scale down reference picture by factor of 2</td></tr>
<tr><td>1h</td><td></td><td>Scale up reference picture by factor of 2</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This bit is derived by driver for Main Profile only. Ignored in Simple and Advanced Profiles. This field is used in both VLD and IT modes. This is derived by driver from the history of RANGERED and RANGEREDFRMsyntax elements (i.e. of forward/preceding reference picture) and those of thecurrent picture. RANGERED is the same as (bPicOverflowBlocks » 3) &1. RANGEREDFRM is the same as (bPicDeblocked » 5) & 1. For the current picture is a B picture, this field represents the state of the forward/preceding reference picture only Driver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent.</td></tr>
</table></td></tr>
<tr><td></td><td>6</td><td><b>RangeReduction Enable</b>

This field specifies whether on-the-fly pixel value range reduction should be performed for the preceding (or forward) reference picture. Along with RangeReductionScale to specify whether scale up or down should be performed. It is not the same value as RANGEREDFRM Syntax Element (PictureParameters bPicDeblocked bit 5) in the Picture Header.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable</td><td>Range reduction is not performed</td></tr>
<tr><td>1h</td><td>Enable</td><td>Range reduction is performed</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This field is for Main Profile only. Simple Profile is always disable, and not applicable to Advanced Profile. This field is used inboth VLD and IT modes. This is derived by driver from the history of RANGERED and RANGEREDFRM syntax elements (i.e. of forward/preceding reference picture) andthose of the current picture. RANGERED is the same as (bPicOverflowBlocks» 3) & 1. RANGEREDFRM is the same as (bPicDeblocked » 5) &1.For the current picture is a B picture, this field represents the state of the forward/preceding reference picture only Driver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of currentpicture coherent.</td></tr>
</table></td></tr>
<tr><td></td><td>5</td><td><b>Reserved</b></td></tr>
</table>

<table>
<tr><th colspan="3">MFD_VC1_LONG_PIC_STATE</th></tr>
<tr><td></td><td>4</td><td><b>Overlap Smoothing Enable Flag</b>

This field is the decoded syntax element OVERLAP in bitstream Indicates if Overlap smoothing is ON at the picture level This field is used in both VLD and IT modes.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable</td><td>to disable overlap smoothing filter</td></tr>
<tr><td>1h</td><td>Enable</td><td>to enable overlap smoothing filter</td></tr>
</table></td></tr>
<tr><td></td><td>3</td><td><b>Secondfield</b>

This flag is set for the second field in field pictures. This field is used in both VLD and IT modes.</td></tr>
<tr><td></td><td>2:1</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>0</td><td><b>VC1 Profile</b>

specifies the bitstream profile. This field is used in both VLD and IT modes.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable</td><td>current picture is in Simple or Main Profile (No need to distinguish Simple and Main Profile)</td></tr>
<tr><td>1h</td><td>Enable</td><td>current picture is in Advanced Profile</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This is required because 128 is added for intra blocks post inverse transform in advanced profile and also to find out if Motion vectors are adjusted or not.</td></tr>
</table></td></tr>
<tr><td>3</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>30:29</td><td><b>CondOver</b>

This field is the decoded syntax element CONDOVER in a bitstream of advanced profile. It controls the overlap smoothing filter operation for an I frame or a BI frame when the picture level qualization step size PQUANT is 8 or lower. This field is used in intel VC1 VLD mode only, not in VC1 and IT modes.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>00b</td><td></td><td>No overlap smoothing</td></tr>
<tr><td>01b</td><td></td><td>Reserved</td></tr>
<tr><td>10b</td><td></td><td>Always perform overlap smoothing filter</td></tr>
<tr><td>11b</td><td></td><td>Overlap smoothing on a per macroblock basis based on OVERFLAGS</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">MFD_VC1_LONG_PIC_STATE</th></tr>
<tr><td></td><td>28:26</td><td><b>PicType (Picture Type)</b>

This field specifies the coding type of the picture according to the Frame Coding Mode. When FCM = 00 | 01 (a Progressive or Interlaced Frame Picture):000 = I001 = P010 = B011 = BI100 = Skipped Other encodings are reserved When FCM = 10 | 11 (a Field Picture)000 = I/I001 = I/P010 = P/I011 = P/P100 = B/B101 = B/BI110 = BI/B111 = BI/BI Although, for a field picture, it is set for a field-pair, but HW will only look at one field state only, and the other field state is don't care. This field is read and qualified with the SecondField flag internally. This field is unique to intel VC1 VLD Long format, and is used in IT mode as well. For VC1 IT mode, driver needs to convert the interface to intel HW VLD Long Format interface.</td></tr>
<tr><td></td><td>25:24</td><td><b>FCM (Frame Coding Mode)</b>

This is the same as the variable FCM defined in VC1.This field must be set to 0 for Simple and Main Profiles This field is unique to intel VC1 VLD Long format, and is used in IT mode as well. For VC1 IT mode, driver needs to convert the interface to intel HW VLD Long Format interface.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>00b</td><td>Disable</td><td>Progressive Frame Picture</td></tr>
<tr><td>01b</td><td>Enable</td><td>Interlaced Frame Picture</td></tr>
<tr><td>10b</td><td></td><td>Field Picture with Top Field First</td></tr>
<tr><td>11b</td><td></td><td>Field Picture with Bottom Field First</td></tr>
</table></td></tr>
<tr><td></td><td>23:21</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>20:16</td><td><b>AltPQuant (Alternative Picture Quantization Value)</b>

This field is identical to the variable ALTPQUANT which is derived from VOPDQUANT configuration in the VC1 standard. This field must be set to 0 for Simple/Main I and BI pictures as VOPDQUANT is not present. This field is used in intel VC1 VLD Long Format mode only, not used in VC1 VLD and IT modes.</td></tr>
<tr><td></td><td>15:13</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>12:8</td><td><b>PQuant (Picture Quantization Value)</b>
<table>
<tr><td>Format:</td><td>U5</td></tr>
</table>
This is the same as the calculated variable PQUANT in VC1 standard where PQuant = PQINDEX, except when QUANTIZER = 0 and PQINDEX> 8, it is given as PQuant = (PQINDEX < 29) ? PQINDEX - 3 : PQINDEX*2 -31This field is used in all picture types (I, P, B and BI) and all operating modes (IT mode and intel and VLD modes).</td></tr>
</table>

<table>
<tr><th colspan="3">MFD_VC1_LONG_PIC_STATE</th></tr>
<tr><td></td><td>7:0</td><td><b>BScaleFactor</b>

BScaleFactor This field is the scale factor for computing Direct-mode motion vectors. It is derived from the variable BFRACTION in the VC1 standard, section 8.4.5.4.There are only 21 valid values corresponding to the 21 encodings of BFRACTION as shown in the table here. Other values are reserved. MSB of this field can be used to determine if BFRACTION is greater than or equal to 1/2, which is used to determine Motion Prediction Type for B pictures. Effectively, condition "BFRACTION >= 1/2" is equivalent to condition "BScaleFactor >= 128".This field is only valid for B pictures. This field is used only in intel VC1 VLD Long format mode, it is not used in VC1 VLD and IT modes. BFRACTIONVLCBFRACTIONBScaleFactor0001/21280011/3850102/31700111/4641003/41921011/5511102/510211100003/515311100014/520411100101/64311100115/621511101001/73711101012/77411101103/711111101114/714811110005/718511110016/722211110101/83211110113/89611111005/816011111017/8224</td></tr>
<tr><td>4</td><td>31:30</td><td><b>Reserved</b>

<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>29:28</td><td><b>UnifiedMvMode (Unified Motion Vector Mode)</b>

This field is a combination of the variables MVMODE and MVMODE2 in the VC1 standard, for parsing Luma MVD from the bitstream. This field is used to signal 1MV vs 4MVallowed (Mixed Mode). This field is also used to signal Q-pel or Half-pel MVD read from the bitstream. The bicubic or bilinear Luma MC interpolation mode is duplicate information from Motion Vector Mode field, and is ignored here. This field is used in intel VC1 VLD Long Format mode only, it is not used in VC1 VLD and IT modes.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>00b</td><td></td><td>Mixed MV, Q-pel bicubic</td></tr><tr><td>01b</td><td></td><td>1-MV, Q-pel bicubic</td></tr><tr><td>10b</td><td></td><td>1-MV half-pel bicubic</td></tr><tr><td>11b</td><td></td><td>1-MV half-pel bilinear</td></tr></table></td></tr>
<tr><td></td><td>27</td><td><b>FourMvSwitch (Four Motion Vector Switch)</b>

This field indicates if 4-MV is present for an interlaced frame P picture. It is identical to the variable 4MVSWITCH (4 Motion Vector Switch) in VC1 standard. This field is used in intel VC1 VLD Long Format mode only, it is not used in VC1 VLD and IT modes.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Disable</td><td>only 1-MV</td></tr><tr><td>1h</td><td>Enable</td><td>1, 2, or 4 MVs</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFD_VC1_LONG_PIC_STATE</th></tr>
<tr><td></td><td>26</td><td><b>FastUVMCFlag (Fast UV Motion Compensation Flag)</b>

This field specifies whether the motion vectors forUV is rounded to half or full pel position. It is identical to the variable FASTUVMC in VC1 standard. This field is used in both VLD and IT modes. It is derived from FASTUVMC = (bPicSpatialResid8 » 4) & 1 in both VLD and IT modes, and should have the same value as Motion Vector ModeLSBit.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td></td><td>no rounding</td></tr>
<tr><td>1h</td><td></td><td>quarter-pel offsets to half/full pel positions</td></tr>
</table></td></tr>
<tr><td></td><td>25</td><td><b>RefFieldPicPolarity (Reference Field Picture Polarity)</b>

This field specifies the polarity of the one reference field picture used for a field P picture. It is derived from the variable REFFIELD defined in VC1 standard and is only valid when one field is referenced (NUMREF = 0) for a field P picture. When NUMREF = 0 and REFFIELD = 0, this field is the polarity of the reference I/P field that is temporally closest; When NUMREF = 0 and REFFIELD = 1, this field is the polarity of the reference I/P field that is the second most temporally closest. The distance is measured based on display order but ignoring the repeated field if present (due to RFF = 1).This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td></td><td>Top (even) field</td></tr>
<tr><td>1h</td><td></td><td>Bottom (odd) field</td></tr>
</table></td></tr>
<tr><td></td><td>24</td><td><b>NumRef (Number of References)</b>

This field indicates how many reference fields are referenced by the current (field) picture. It is identical to the variable NUMREF in the VC1 standard. This field is only valid for field P picture (FCM = 10 | 11).This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td></td><td>One field referenced</td></tr>
<tr><td>1h</td><td></td><td>Two fields referenced</td></tr>
</table></td></tr>
<tr><td></td><td>23:20</td><td><b>BwdRefDist (Reference Distance)</b>

This field is valid only in B field pictures giving the value of BRFD. The field is ignored in P Picture. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.</td></tr>
</table>

MFD_VC1_LONG_PIC_STATE

19:16 **FwdRefDist (Reference Distance)**

Format:	U4

This field is the number of frames between the current frame and its reference frame. It is derived from the syntax element REFDIST (P Reference Distance) in the VC1 standard. 0 means that the previous frame is the reference frame. It has the same value as of FRFD for both P and B field pictures. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.

Value	Name
[0, 15]	

15:12 **Reserved**

Access:	RO
Format:	MBZ

11:10 **ExtendedDMVRange (Extended Differential Motion Vector Range Flag)**

This field specifies the differential motion vector range in interlaced pictures. It is equivalent to the variable DMVRANGE in the VC1 standard. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.

Value	Name	Description
00b		No extended range
01b		Extended horizontally
10b		Extended vertically
11b		Extended in both directions

9:8 **ExtendedMVRange (Extended Motion Vector Range Flag)**

This field specifies the motion vector range in quarter-pel or half-pel modes. It is equivalent to the variable MVRANGE in the VC1 standard. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes

Value	Name	Description
00b		[-256, 255] x [-128, 127]
01b		512, 511] x [-256, 255]
10b		[-2048, 2047] x [-1024, 1023]
11b		[-4096, 4095] x [-2048, 2047]

<table>
<tr><th colspan="3">MFD_VC1_LONG_PIC_STATE</th></tr>
<tr><td></td><td>7:4</td><td><b>AltPQuantEdgeMask (Alternative Picture Quantization Edge Mask)</b>

This field is a bit mask for the four edges in clock-wise order, indicating whether AltPQuant is used for the edge macroblocks. It is derived based on the following variables DQUANT, DQUANTFRM, DQPROFILE, DQSBEDGE, DQDBEDGE, and DQBILEVEL defined in the VC1 standard, as shown in Error! Reference source not found. This field is valid only if AltPQuantConfig is 01. Bit 0: Left picture edge macroblocks Bit 1: Top picture edge macroblocks Bit 2: Right picture edge macroblocks Bit 3: Bottom picture edge macroblocks This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.</td></tr>
<tr><td></td><td>3:2</td><td><b>AltPQuantConfig (Alternative Picture Quantization Configuration)</b>

This field specifies the way AltPQuant is used in the picture. It determines how to compute the macroblock quantizer step size, MQUANT. It is derived based on the following variables DQUANT, DQUANTFRM, DQPROFILE, DQSBEDGE, DQDBEDGE, and DQBILEVEL defined in the VC1 standard, as shown in Error! Reference source not found. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and modes.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>00b</td><td></td><td>AltPQuant not used</td></tr>
<tr><td>01b</td><td></td><td>AltPQuant is used and applied to edge macroblocks only</td></tr>
<tr><td>10b</td><td></td><td>MQUANT is encoded in macroblock layer</td></tr>
<tr><td>11b</td><td></td><td>AltPQuant and PQuant are selected on macroblock basis</td></tr>
</table></td></tr>
<tr><td></td><td>1</td><td><b>HalfQP</b>

This field is used for inverse quantization of AC coefficients. It is valid only when PQuant is used. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.</td></tr>
<tr><td></td><td>0</td><td><b>PQuantUniform</b>

Indicating if uniform quantization applies to the picture. It is used for inverse quantization of the AC coefficients.QUANTIZER001123PQUANTIZER --01--PQINDEX>=9<=8----PQuantUniform010201ImplicitQuantizer = 0, and PQuantUniform = 0 is used to represent 2 cases : 1) QUANTIZER=01 and PQUANTIZER=0; and 2) QUANTIZER = 10b.ImplicitQuantizer = 0, and PQuantUniform =1 is used to represent 2 cases : 1) QUANTIZER=01 and PQUANTIZER=1; and 2)QUANTIZER = 11bThis field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td></td><td>Non-uniform</td></tr>
<tr><td>1h</td><td></td><td>Uniform</td></tr>
</table></td></tr>
</table>

MFD_VC1_LONG_PIC_STATE

5 | 31 | **BitplanePresentFlag (Bitplane Buffer Present Flag)**

This field indicates whether the bitplane buffer is present for the picture. If set, at least one of the fields listed in bits 22:16 is coded in non-raw mode, and Bitplane Buffer Base Address field in the VC1_BSD_BUF_BASE_STATE command points to the bitplane buffer. Otherwise, all the fields that are applicable for the current picture in bits 22:16 must be coded in raw mode. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.

Value	Name	Description
0h		bitplane buffer is not present
1h		bitplane buffer is present

30 | **ForwardMbRaw**

This field indicates whether the FORWARDMB field is coded in raw or non-raw mode. This field is only valid when PictureType is B. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.

Value	Name	Description
0h		non-raw mode
1h		raw mode

29 | **MvTypeMbRaw**

This field indicates whether the MVTYPREMB field is coded in raw or non-raw mode. This field is only valid when PictureType is P. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.

Value	Name	Description
0h		Non-Raw Mode
1h		Raw Mode

28 | **SkipMbRaw**

This field indicates whether the SKIPMB field is coded in raw or non-raw mode. This field is only valid when PictureType is P or B.0 = non-raw mode1 = raw mode This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.

Value	Name	Description
0h	Disable	Non-Raw Mode
1h	Enable	Raw Mode

27 | **DirectMbRaw**

This field indicates whether the DIRECTMB field is coded in raw or non-raw mode. This field is only valid when PictureType is P or B. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.

Value	Name	Description
0h		Non-Raw Mode
1h		Raw Mode

<table>
<tr><th colspan="3">MFD_VC1_LONG_PIC_STATE</th></tr>
<tr><td></td><td>26</td><td><b>OverflagsRaw</b>

This field indicates whether the OVERFLAGS field is coded in raw or non-raw mode. This field is only valid when PictureType is I or BI. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td></td><td>Non-Raw Mode</td></tr>
<tr><td>1h</td><td></td><td>Raw Mode</td></tr>
</table></td></tr>
<tr><td></td><td>25</td><td><b>AcPredRaw</b>

This field indicates whether the ACPRED field is coded in raw or non-raw mode. This field is only valid when PictureType is I or BI. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable</td><td>Non-Raw Mode</td></tr>
<tr><td>1h</td><td>Enable</td><td>Raw Mode</td></tr>
</table></td></tr>
<tr><td></td><td>24</td><td><b>FieldTxRaw</b>

This field indicates whether the FIELDTX field is coded in raw or non-raw mode. This field is only valid when PictureType is I or BI. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable</td><td>Non-Raw Mode</td></tr>
<tr><td>1h</td><td>Enable</td><td>Raw Mode</td></tr>
</table></td></tr>
<tr><td></td><td>23</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

MFD_VC1_LONG_PIC_STATE

	22:20	**MvTab (Motion Vector Table)** Format: U3 This field specifies which motion vector table(s) is (are) used for motion vector (differential) decoding in a P or B picture. This field is the combination of the variables MVTAB and IMVTAB in the VC1 standard. Two bits are defined for progressive frame pictures; And two or three bits are defined for interlaced field/frame pictures depending on NUMREF and P or B picture types. This field is valid for P and B pictures. It is not valid for I pictures. For P or B progressive frame pictures0 = Motion Vector Differential VLD Table 01 = Motion Vector Differential VLD Table 12 = Motion Vector Differential VLD Table 23 = Motion Vector Differential VLD Table 3The other encodings are reserved For P interlace field pictures with NUMREF = 0 or P/B interlace frame pictures0 = 1-Reference Table 01 = 1-Reference Table 12 = 1-Reference Table 23 = 1-Reference Table 3The other encodings are reserved For P interlace field picture with NUMREF = 1 or B interlaced field pictures0 = 2-Reference Table 01 = 2-Reference Table 12 = 2-Reference Table 23 = 2-Reference Table 34 = 2-Reference Table 45 = 2-Reference Table 56 = 2-Reference Table 67 = 2-Reference Table 7The other encodings are reserved This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.
	19:18	**FourMvBpTab (4-MV Block Pattern Table)** This field specifies which table is used to decode the 4-MV block pattern (4MVBP) syntax element in 4-MV macroblocks. It is identical to the variables 4MVBPTAB in the VC1 standard, section 9.1.1.37. This field is valid only in interlace frame P, B pictures, or interlace field P, B pictures. It is not valid for I picture. For interlace field P and B pictures, it is only valid if UnifiedMvMode is equal to Mixed-MV Type. For interlace frame P picture, it is only valid if FourMvSwitch is 1.For interlace frame B picture, it is always valid.0 = 4MVBP Table 01 = 4MVBP Table 12 = 4MVBP Table 23 = 4MVBP Table 3This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.
	17:16	**TwoMvBpTab (2MV Block Pattern Table)** This field specifies which table is used to decode the 2MV block pattern (2MVBP) syntax element in 2MV field macroblocks. It is identical to the variables 2MVBPTAB in the VC1 standard, section 9.1.1.36. This field is valid only in interlace frame P/B pictures. It is not valid for I picture, nor for interlace field P or B pictures.0 = 2MVBP Table 01 = 2MVBP Table 12 = 2MVBP Table 23 = 2MVBP Table 3This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.
	15:14	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">MFD_VC1_LONG_PIC_STATE</th></tr>
<tr><td></td><td>13:12</td><td><b>TransType (Picture-level Transform Type)</b>
<table><tr><td>Format:</td><td>U2</td></tr></table>This field specifies the Transform Type at picture level. It is identical to the variable TTFRM in the VC1 standard, section 7.1.1.41. This field is only valid when TransTypeMbFlag is 1. Otherwise, it is reserved and MBZ. This field is set to 00 when VSTRANSFORM is 0 in the entry point layer.00 = 8x8 Transform01 = 8x4 Transform10 = 4x8 Transform11 = 4x4 Transform This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.</td></tr>
<tr><td></td><td>11</td><td><b>TransTypeMbFlag (Macroblock Transform Type Flag)</b>
This field indicates whether Transform Type is fixed at picture level or variable at macroblock level. It is identical to the variable TTMBF in the VC1 standard, section 7.1.1.40.This field is set to 1 when VSTRANSFORM is 0 in the entry point layer. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td></td><td>variable transform type in macroblock layer</td></tr><tr><td>1h</td><td></td><td>use picture level transform type TransType</td></tr></table></td></tr>
<tr><td></td><td>10:8</td><td><b>MbModeTab (Macroblock Mode Table)</b>
This field signals which code table is used to decode the macroblock mode syntax element (MBMODE) in the macroblock layer in a P or B picture. This field is identical to the variables MBMODETAB in the VC1 standard, section 9.1.1.33. This field is valid for interlace frame P, B picture and interlace field P, B picture. It is not valid for I picture, nor progressive frame P, B pictures. Two bits are defined for interlace frame P, B pictures; And three bits are defined for interlaced field P, B pictures. Two bits are defined for interlace frame P, B pictures. There are two set of code tables selected based on if UnifiedMvMode is equal to 4-MV Type or not. 0 = Code Table 01 = Code Table 12 = Code Table 23 = Code Table 3Other encodings are invalid Three bits are defined for interlace field P, B pictures. There are two set of code tables selected based on if UnifiedMvMode is equal to Mixed-MV Type or not. 0 = Code Table 01 = Code Table 12 = Code Table 23 = Code Table 34 = Code Table 45 = Code Table 56 = Code Table 67 = Code Table 7This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.</td></tr>
<tr><td></td><td>7:6</td><td><b>TransAcY (Picture-level Transform Luma AC Coding Set Index, TRANSACTABLE2</b></td></tr>
<tr><td></td><td>5:4</td><td><b>TransAcUV (Picture-level Transform Chroma AC Coding Set Index, TRANSACTABLE)</b>
This field, together with PQINDEX, specifies which intra AC coding set to be used for decoding the non-zero AC coefficients in a coded luma (Y) block. This field is the combination of the variables TRANSACFRM and TRANSACFRM2 in the VC1 standard. For I pictures, TransAcY is the same as TRANSACFRM2. For other pictures, it is the same as TRANSACFRM, and therefore must be programmed to be the same as TransAcUV. This field is valid for all picture types.0 = Coding set index 01 = Coding set index 12 = Coding set index 23 is invalid This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.</td></tr>
</table>

<table>
<tr><th colspan="3">MFD_VC1_LONG_PIC_STATE</th></tr>
<tr><td></td><td>3</td><td><b>TransDcTab (Intra Transform DC Table)</b>

This field specifies whether the low motion tables or the high motion tables are used to decode the Transform DC coefficients in intra-coded blocks. This field is identical to the variable TRANSDCTAB in the VC1 standard, section 8.1.1.2. This field is valid for all picture types. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td></td><td>The high motion tables</td></tr>
<tr><td>1h</td><td></td><td>The low motion tables</td></tr>
</table></td></tr>
<tr><td></td><td>2:0</td><td><b>CbpTab (Coded Block Pattern Table)</b>

This field specifies the table used to decode the CBPCY syntax element for each coded macroblock in P and B pictures. This field is combination of the variable CBPTAB for P and B frame pictures and the variable ICBPTAB in interlace field P, B pictures and interlace frame P, B pictures in the VC1 standard (Table 52 and Table 102).This field is reserved and MBZ for I or BI pictures as I only has a fixed table.000 = Table 0 (Table 169 for P, B frames or Table 124 otherwise)001 = Table 1 (Table 170 for P, B frames or Table 125 otherwise)010 = Table 2 (Table 171 for P, B frames or Table 126 otherwise)011 = Table 3 (Table 172 for P, B frames or Table 127 otherwise)100 = Table 4 (Table 128 for interlace field/frame P, B pictures)101 = Table 5 (Table 129 for interlace field/frame P, B pictures)110 = Table 6 (Table 130 for interlace field/frame P, B pictures)111 = Table 7 (Table 131 for interlace field/frame P, B pictures)This field is unique to intel VC1 VLD Long format mode, and is not used in IT and VC1 modes.</td></tr>
</table>

MFD_VC1_SHORT_PIC_STATE

MFD_VC1_SHORT_PIC_STATE

Source: VideoCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_VC1_SHORT_PIC_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 0h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 0003h Excludes DWord (0,1) Format: =n
1	31:24	**Reserved** Access: RO Format: MBZ

MFD_VC1_SHORT_PIC_STATE

	23:16	**Picture Height** Format: U8-1 This field indicates the height of the picture in unit of macroblocks. For example, for a 1920x1080 frame picture, PictureHeightInMBs equals 68 (1080 divided by 16, and rounded up, i.e. effectively specified as 1088 instead). This field is used in VLD and IT modes. Note: Even though the Advanced Profile allows frame dimensions (width, height) to not be aligned to macroblock boundary, it doesn't affect the bitstream decoding. And it is preferable to use 'intermediate buffer' that is macroblock aligned for decoding. In order to simplify the out-of-bound reference pixel access, the out-of-bound extrapolation rule in VC1 spec can be used to expand the expected decoded frame to the intermediate buffer dimension.
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**Picture Width** Format: U8-1 This field indicates the width of the picture in unit of macroblocks. For example, for a 1920x1080 frame picture, PictureWidthInMBs equals 120 (1920 divided by 16).This field is used in VLD and IT modes.
2	31:24	**Bitplane Buffer Pitch Minus 1** Format: U8-1 Specifies the bitplane buffer pitch in (#Bytes - 1). Bitplane buffer is a linear buffer. It is needed only when the bitplane is not encoded as raw, and therefore is present in the header explicitly. In VC1 Long Format, it is written by an application and later read by the HW. In VC1 Long Format, it is written by an application, and later read by the HW. But in VC1 Short Format, it is written and read by H/W only. This field is specified for better performance: The pitch must be equal to PictureWidthInMBs/2. For VC1 Long Format: The pitch must be equal to PictureWidthInMBs/2. For VC1 Short Format : If Pic Width is less than or equal to 2K pixels, bitplane pitch is set to 64 (one cacheline; programmed as 63) bytes per MB row. If Pic Width is greater than 2K pixels, bitplane pitch is set to 128 (two cachelines; programmed as 127) bytes per MB row. This field is not used in IT mode, used in VLD mode only.For VC1 Short Format, the bitplane specification is between H/W and Driver only. For Long Format, application is responsible for allocation with the driver.
	23	**Interpolation Rounder Control** Used only in MC operation. This field specifies the rounding control value used in interpolation operation of motion prediction process. Note: This bit field is taken from bRcontrol in PictureParameters data structure This field is used in VLD and IT modes.

<table>
<tr><th colspan="3">MFD_VC1_SHORT_PIC_STATE</th></tr>
<tr><td></td><td>22:20</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>19:16</td><td><b>Motion Vector Mode</b>

This field indicates one of the following motion compensation interpolation modes for P and B pictures. The MC interpolation modes apply to prediction values of luminance blocks and are always in quarter-sample. For chrominance blocks, it always performs bilinear interpolation with either half-pel or quarter-pel precision.0XX0 = Chroma Quarter -pel + Luma bicubic. (can only be 1MV)0XX1 = Chroma Half-pel + Luma bicubic. (can be 1MV or 4MV)1XX0 = Chroma Quarter -pel + Luma bilinear. (can only be 1MV)1XX1 = Chroma Half-pel + Luma bilinearNote: Bits 19:16 are taken from bMVprecisionAndChromaRelation in PictureParameters data structure.Bit 19 of Motion Vector Mode = 1 for Luma Bilinear MC; = 0 for Luma Bicubic MCBit 16 of Motion Vector Mode = 1 for half-sample Chroma motion = 0 for quarter-sample Chroma motion.This field is used in both VLD and IT modes.Before the polarity of Chroma Half-pel or Q-pel is reversed from Spec, now I have fixed it to match with VC1 Spec.</td></tr>
<tr><td></td><td>15</td><td><b>DmvSurfaceValid</b>

Indicated when the DMV read surface is valid. This surface stored the direct motion vectors. This field is set for B pictures that can refer to a previous P picture for DMV. If there is an I-picture before a B (in decoding order) then this field is not set (as a result, zero's DMV's will be assumed while decoding the B picture. That is, there is no explicit DMV buffer for an I-picture). This field is not used in IT mode, used in VLD mode only.</td></tr>
<tr><td></td><td>14:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11</td><td><b>VC1 Profile</b>

specifies the bitstream profile. Note: This is required because 128 is added for intra blocks post inverse transform in advanced profile and also to find out if Motion vectors are adjusted or not. This field is used in both VLD and IT modes.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td><b>[Default]</b></td><td>current picture is in Simple or Main Profile (No need to distinguish Simple and Main Profile)</td></tr><tr><td>1h</td><td></td><td>current picture is in Advanced Profile</td></tr></table></td></tr>
<tr><td></td><td>10:6</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFD_VC1_SHORT_PIC_STATE</th></tr>
<tr><td></td><td>5</td><td><b>Backward Prediction Present Flag</b>

Note : a B picture that only uses forward prediction may have this flag set to 1 as well. Driver may still need to provide a valid reference picture index. This field is used in both VC1 VLD mode and IT mode. It is the same parameter as bPicBackwardPrediction in VC1 spec. The Intra Picture Flag, Backward Prediction Present Flag and RefPicFlag are used to derive the picture type, as specified in PTYPE for a frame, and in FPTYPE for a field, in VC1 VLD and IT mode.</td></tr>
<tr><td></td><td>4</td><td><b>Intra Picture Flag</b>

This field is used in both VC1 VLD mode and IT mode. It is the same parameter as bPicIntra in VC1 spec. The Intra Picture Flag, Backward Prediction Present Flag and RefPicFlag are used to derive the picture type, as specified in PTYPE for a frame, and in FPTYPE for a field, in VC1 VLD and IT mode.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td></td><td>entire picture can have a mixture of intra and inter MB type or just inter MB type.</td></tr>
<tr><td>1h</td><td></td><td>entire picture is coded in intra MB type</td></tr>
</table></td></tr>
<tr><td></td><td>3</td><td><b>SecondField</b>

This flag is set for the second field in field pictures. This field is used in both VLD and IT modes.</td></tr>
<tr><td></td><td>2</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>1:0</td><td><b>Picture Structure</b>

This field is used in both VC1 VLD mode and IT mode. It is the same parameter as bPicStructure in VC1 spec. The Picture Structure and Progressive Pic Type are used to derive the picture structure as specified in FCM, in VC1 VLD and IT mode.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>01b</td><td></td><td>top field (bit 0)</td></tr>
<tr><td>10b</td><td></td><td>bottom field (bit 1)</td></tr>
<tr><td>11b</td><td></td><td>frame (both fields are present)</td></tr>
<tr><td>00b</td><td></td><td>illegal</td></tr>
</table></td></tr>
<tr><td>3</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

MFD_VC1_SHORT_PIC_STATE

30 **Overlap Smoothing Enable Flag**

This field is the decoded syntax element OVERLAP in bitstream Indicates if Overlap smoothing is ON at the picture level This field is used in both VLD and IT modes

Value	Name	Description
0h	Disable	to disable overlap smoothing filter
1h	Enable	to enable overlap smoothing filter

29 **Range Reduction Scale**

Access:	None

This field specifies whether the reference picture pixel values should be scaled up or scaled down on-the-fly, if RangeReduction is Enabled. NOTE: This bit is derived by driver for Main Profile only. Ignored in Simple and Advanced Profiles. This field is used in both VLD and IT modes. This is derived by driver from the history of RANGERED and RANGEREDFRM syntax elements (i.e. of forward/preceding reference picture) and those of the current picture. RANGERED is the same as (bPicOverflowBlocks » 3) & 1. RANGEREDFRM is the same as (bPicDeblocked » 5) & 1. For the current picture is a B picture, this field represents the state of the forward/preceding reference picture only Driver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent.

Value	Name	Description
0h	Disable **[Default]**	Scale down reference picture by factor of 2
1h	Enable	Scale up reference picture by factor of 2

28 **Range Reduction Enable**

This field specifies whether on-the-fly pixel value range reduction should be performed for the preceding (or forward) reference picture. Along with RangeReductionScale to specify whether scale up or down should be performed. It is not the same value as RANGEREDFRM Syntax Element(PictureParameters bPicDeblocked bit 5) in the Picture Header. This field is for Main Profile only. Simple Profile is always disable, and not applicable to Advanced Profile. This field is used in both VLD and IT modes. This is derived by driver from the history of RANGERED and RANGEREDFRM syntax elements (i.e. of forward/preceding reference picture) and those of the current picture. RANGERED is the same as (bPicOverflowBlocks » 3) & 1. RANGEREDFRM is the same as (bPicDeblocked » 5) & 1.For the current picture is a B picture, this field represents the state of the forward/preceding reference picture only Driver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent.

Value	Name	Description
0h	Disable **[Default]**	Range reduction is not performed
1h	Enable	Range reduction is performed

MFD_VC1_SHORT_PIC_STATE

27:24 **Reserved**

Access:	RO
Format:	MBZ

23:22 **Progressive Pic Type**

This field is used in both VC1 VLD mode and IT mode. It is the same parameter as bPicExtrapolation in VC1 spec. The Picture Structure and Progressive Pic Type are used to derive the picture structure as specified in FCM, in VC1 VLD and IT mode.

Value	Name	Description
0		progressive only picture
1		progressive only picture
2		interlace picture (frame-interlace or field-interlace)
3		illegal

21 **Reserved**

Access:	RO
Format:	MBZ

20:16 **P-Pic Ref Distance**

Access:	None

This element defines the number of frames between the current frame and the reference frame. It is the same as the REFDIST SE in VC1 interlaced field picture header. It is present if the entry-level flag REFDIST_FLAG == 1, and if the picture type is not one of the following types: B/B, B/BI, BI/B, BI/BI. If the entry level flag REFDIST_FLAG == 0, REFDIST shall be set to the default value of 0. This field is used in VC1 VLD mode only, not used in IT and intel VC1 VLD Long Format modes.

Value	Name
0-16	unsigned integer
0h	**[Default]**

15:14 **QUANTIZER**

Value	Name	Description
00b		implicit quantizer at frame level
01b		explicit quantizer at frame level, and use PQUANTIZER SE to specify uniform or non-uniform
10b		explicit quantizer, and non-uniform quantizer for all frames
11b		explicit quantizer, and uniform quantizer for all frames

<table>
<tr><th colspan="3">MFD_VC1_SHORT_PIC_STATE</th></tr>
<tr><td></td><td>13</td><td><b>MULTIRES Present Flag (for Simple/Main Profile only)</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td></td><td>RESPIC Parameter is present in the picture header</td></tr>
<tr><td>1h</td><td></td><td>RESPIC Parameter is present in the picture header</td></tr>
</table></td></tr>
<tr><td></td><td>12</td><td><b>SYNCMARKER Present Flag (for Simple/Main Profile only)</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td></td><td>Bitstream for Simple and Main Profile has no sync marker</td></tr>
<tr><td>1</td><td></td><td>Bitstream for Simple and Main Profile may have sync marker(s)</td></tr>
</table></td></tr>
<tr><td></td><td>11</td><td><b>RANGERED Present Flag (for Simple/Main Profile only)</b>
It is needed for Picture Header Parsing. Driver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td></td><td>Range Reduction Parameter (RANGEREDFRM) is not present in the picture header</td></tr>
<tr><td>1</td><td></td><td>Range Reduction Parameter (RANGEREDFRM) is present in the picture header.</td></tr>
</table></td></tr>
<tr><td></td><td>10:8</td><td><b>MAXBFRAMES</b>
Number of consecutive B Frames.</td></tr>
<tr><td></td><td>7</td><td><b>PANSCAN Present Flag</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td></td><td>Pan Scan Parameters are not present in the picture header</td></tr>
<tr><td>1</td><td></td><td>Pan Scan Parameters are present in the picture header</td></tr>
</table></td></tr>
<tr><td></td><td>6</td><td><b>REFDIST_FLAG</b>
For header parsing REFDIST. This is used in VC1 VLD mode only, not used in IT and intel VC1 VLD modes.</td></tr>
<tr><td></td><td>5</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>4</td><td><b>FastUVMCFlag (Fast UV Motion Compensation Flag)</b>
This field specifies whether the motion vectors for UV is rounded to half or full pel position. It is identical to the variable FASTUVMC in VC1 standard. This field is used in both VLD and IT modes. It is derived from FASTUVMC = (bPicSpatialResid8 » 4) & 1 in both VLD and IT modes, and should have the same value as Motion Vector ModeLSBit.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td></td><td>no rounding</td></tr>
<tr><td>1h</td><td></td><td>quarter-pel offsets to half/full pel positions</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">MFD_VC1_SHORT_PIC_STATE</th></tr>
<tr><td></td><td>3</td><td><b>EXTENDED_MV Present Flag</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td></td><td>Extended_MV is not present in the picture header</td></tr>
<tr><td>1h</td><td></td><td>Extended_MV is present in the picture header</td></tr>
</table></td></tr>
<tr><td></td><td>2:1</td><td><b>DQUANT</b>
<table>
<tr><td>Access:</td><td>None</td></tr>
<tr><td>Format:</td><td>U2</td></tr>
</table>
Use for Picture Header Parsing of VOPDUANT elements
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td><b>[Default]</b></td><td></td></tr>
<tr><td>00b</td><td></td><td>no VOPDQUANT elements; Quantizer cannot vary in frame, same quantization step size PQUANT is used for all MBs in the frame</td></tr>
<tr><td>01b</td><td></td><td>refer to VC1 Spec. for all the MB position dependent quantizer selection</td></tr>
<tr><td>10b</td><td></td><td>The macroblocks located on the picture edge boundary shall be quantized with ALTPQUANT while the rest of the macroblocks shall be quantized with PQUANT.</td></tr>
<tr><td>11b</td><td>Reserved</td><td></td></tr>
</table></td></tr>
<tr><td></td><td>0</td><td><b>VSTRANSFORM flag</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable</td><td>variable-sized transform coding is not enabled</td></tr>
<tr><td>1h</td><td>Enable</td><td>variable-sized transform coding is enabled</td></tr>
</table></td></tr>
<tr><td>4</td><td>31:29</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">MFD_VC1_SHORT_PIC_STATE</th></tr>
<tr><td></td><td>28:24</td><td><b>BFraction Enumeration</b>

This field is the scale factor for computing Direct-mode motion vectors. It is derived from the variable BFRACTION in the VC1standard, section 8.4.5.4.There are only 21 valid values corresponding to the 21encodings of BFRACTION as shown in the table here. Other values are reserved. The VLD decoded value of BFRACTION (from the picture header) is mapped into an enum value from 0 to 20.(??? MSB of this field can be used to determine if BFRACTION is greater than or equal to 1/2, which is used to determine MotionPrediction Type for B pictures. Effectively, condition "BFRACTION >= 1/2" is equivalent to condition "ScaleFactor >= 128". ??? How can the enum replicate this feature ???)This field is only valid for B pictures. This field is used only in VC1 VLD mode, it is not used in Intel VC1 VLD Long Format mode and IT mode.
BFRACTIONVLCBFRACTIONEnum0001/200011/310102/320111/431003/441011/551102/5611100003/5711100014/5811100101/6911100115/61011101001/71111101012/71211101103/71311101114/71411110005/71511110016/71611110101/81711110113/81811111005/81911111017/8201111111BIPic Indicator31 (optional)</td></tr>
<tr><td></td><td>23:9</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>8</td><td><b>4MV Allowed Flag</b></td></tr>
<tr><td></td><td>7</td><td><b>POSTPROC Flag</b></td></tr>
<tr><td></td><td>6</td><td><b>PULLDOWN</b></td></tr>
<tr><td></td><td>5</td><td><b>INTERLACE</b></td></tr>
<tr><td></td><td>4</td><td><b>TFCNTRFLAG</b></td></tr>
<tr><td></td><td>3</td><td><b>FINTERFLAG</b></td></tr>
<tr><td></td><td>2</td><td><b>REFPIC Flag</b>

For a BI picture, REFPIC flag must set to 0For I and P picture, REFPIC flag must set to 0.For a B picture, REFPIC flag must set to 0, except for a B-field in interlaced field mode which can be 0 or 1 (e.g. the top B field can be used as a reference for decoding its corresponding bottom B-field in a field pair).In VLD mode, this flag cannot be used as an optimization signaling for an I or P picture that is not used as a reference picture. This field is used in both VC1 VLD mode and IT mode. It is the same parameter as bPicDeblockConfined[bit2] in VC1 spec. The Intra Picture Flag, Backward Prediction Present Flag and RefPicFlag are used to derive the picture type, as specified in PTYPE for a frame, and in FPTYPE for a field, in VC1 VLD and IT mode.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td></td><td>the current picture after decoded, will never used as a reference picture</td></tr><tr><td>1h</td><td></td><td>the current picture after decoded, will be used as a reference picture later</td></tr></table></td></tr>
<tr><td></td><td>1</td><td><b>PSF</b></td></tr>
</table>

<table>
<tr><th colspan="3">MFD_VC1_SHORT_PIC_STATE</th></tr>
<tr><td></td><td>0</td><td><b>EXTENDED_DMV Present Flag</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td><b>[Default]</b></td><td>Extended_DMV is not present in the picture header</td></tr>
<tr><td>1h</td><td></td><td>Extended_DMV is present in the picture header</td></tr>
</table>
</td></tr>
</table>

MFD_VP8_BSD_OBJECT

MFD_VP8_BSD_OBJECT

Source:	VideoCS
Length Bias:	2

The MFD_VP8_BSD_OBJECT command is the only primitive command for the VP8 Decoding Pipeline. The Partitions of the bitstream is loaded as indirect data object. Before issuing a MFD_VP8_BSD_OBJECT command, all VP8 frame level states of the MFD Engine need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of a MFD_VP8_BSD_OBJECT command. Context switch interrupt is not supported by this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_VP8_BSD_OBJECT Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VP8_DEC Format: OpCode
	23:21	**subOpcodeA** Default Value: 1h Format: OpCode
	20:16	**subOpcodeB** Default Value: 8h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 14h Excludes DWord (0,1) Format: =n
1	31:21	**Reserved** Access: RO Format: MBZ
	20:16	**Partition0 CPBAC Entropy Count** Pass the Partition0 CPBAC State to HW.Max value is 24.

MFD_VP8_BSD_OBJECT

	15:8	**Partition0 CPBAC Entropy Range** Pass the Partition0 CPBAC State to HW.
	7:6	**Reserved** Access: RO Format: MBZ
	5:4	**Coded Num of Coeff Token Partitions** Num of Partitions = 2^CodedNumCoeffTokenParititons.0 = 1 Partition only1 = 2 Partitions2 = 4 Partitions3 = 8 Partitions are present in the bitstream.
	3	**Reserved** Access: RO Format: MBZ
	2:0	**Partition0 First MB Bit Offset from Frame Header** Allow HW to jump to the location in the bitstream where per MB information starts in the Partition0.
2	31:24	**Partition0 CPBAC Entropy Value** Pass the Partition0 CPBAC State to HW.
	23:0	**Reserved** Access: RO Format: MBZ
3	31:24	**Reserved** Access: RO Format: MBZ
	23:0	**Indirect Partition0 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 0 length + 1) in bytes
4	31:0	**Indirect Partition0 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
5	31:24	**Reserved** Access: RO Format: MBZ

MFD_VP8_BSD_OBJECT

	23:0	**Indirect Partition1 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 1 length + 1) in bytes
6	31:0	**Indirect Partition1 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
7	31:24	**Reserved** Access: RO Format: MBZ
	23:0	**Indirect Partition2 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 2 length + 1) in bytes
8	31:0	**Indirect Partition2 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
9	31:24	**Reserved** Access: RO Format: MBZ
	23:0	**Indirect Partition3 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 3 length + 1) in bytes

MFD_VP8_BSD_OBJECT

10	31:0	**Indirect Partition3 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
11	31:24	**Reserved** Access: RO Format: MBZ
	23:0	**Indirect Partition4 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 4 length + 1) in bytes
12	31:0	**Indirect Partition4 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
13	31:24	**Reserved** Access: RO Format: MBZ
	23:0	**Indirect Partition5 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 5 length + 1) in bytes
14	31:0	**Indirect Partition5 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
15	31:24	**Reserved** Access: RO Format: MBZ

MFD_VP8_BSD_OBJECT

	23:0	**Indirect Partition6 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 6 length + 1) in bytes
16	31:0	**Indirect Partition6 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
17	31:24	**Reserved** Access: RO Format: MBZ
	23:0	**Indirect Partition7 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 7 length + 1) in bytes
18	31:0	**Indirect Partition7 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
19	31:24	**Reserved** Access: RO Format: MBZ
	23:0	**Indirect Partition8 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition. **Programming Notes** This needs to be set to the (actual Partition 8 length + 1) in bytes

<table>
<tr><th colspan="3">MFD_VP8_BSD_OBJECT</th></tr>
<tr><td>20</td><td>31:0</td><td><b>Indirect Partition8 Data Start Offset</b>
This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.</td></tr>
<tr><td>21</td><td>31</td><td><b>Concealment Method</b>
This field specifies the method used for concealment when error is detected.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Intra 16x16 Prediction</td><td>A copy from the current picture is performed using Intra 16x16 Prediction method.</td></tr>
<tr><td>1</td><td>Inter P Copy</td><td>A copy from collocated macroblock location is performed from the concealment reference indicated by the ConCeal_Pic_Id field.</td></tr>
</table></td></tr>
<tr><td></td><td>30:18</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>17:16</td><td><b>Conceal_Pic_Id (Concealment Picture ID)</b>
<table>
<tr><td>Exists If:</td><td>[Concealment Method] == 1</td></tr>
</table>
This field identifies the picture in the reference list to be used for concealment. This field is only valid if Concealment Method is Inter P Copy.00 - Last Decoded Picture01 - Golden Reference Picture02 - Alternate Reference Picture03 - User provided Reference Picture</td></tr>
<tr><td></td><td>15</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>14</td><td><b>BSD Premature Complete Error Handling</b>
It occurs in situation where the decode is completed but there are still data in the bitstream.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error handling</td></tr>
<tr><td>1</td><td>Set the interrupt to the driver (provide MMIO registers for MB address R/W)</td></tr>
</table></td></tr>
<tr><td></td><td>13</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>12</td><td><b>MPR Error (MV out of range) Handling</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error handling</td></tr>
<tr><td>1</td><td>Set the interrupt to the driver (provide MMIO registers for MB address R/W)</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">MFD_VP8_BSD_OBJECT</th></tr>
<tr><td></td><td>11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Entropy Error Handling</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error handling</td></tr><tr><td>1</td><td>Set the interrupt to the driver (provide MMIO registers for MB address R/W)</td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>8</td><td><b>MB Header Error Handling</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error handling</td></tr><tr><td>1</td><td>Set the interrupt to the driver (provide MMIO registers for MB address R/W)</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

MFX_AVC_DIRECTMODE_STATE

MFX_AVC_DIRECTMODE_STATE

Source:	VideoCS
Length Bias:	2

This is a picture level command and is issued once per picture. All DMV buffers are treated as standard media surfaces, in which the lower 6 bits are used for conveying surface states. Current Pic POC number is assumed to be available in POCList[32 and 33] of the MFX_AVC_DIRECTMODE_STATE Command. This command is only valid in the AVC decoding in VLD and IT modes, and AVC encoder mode. The same command supports both Long and Short AVC Interface. The DMV buffers are not required to be programmed for encoder mode.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_SINGLE_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_COMMON Format: OpCode
	23:21	**SubOpcodeA** Default Value: 0h Format: OpCode
	20:16	**SubOpcodeB** Default Value: 2h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 0045h Excludes DWord (0,1) Format: =n
1..32	1023:0	**Direct MV Buffer for Reference Frame 0 to 15 - Base Address** Format: SplitBaseAddress64ByteAligned[16] This field is for the Pre-Deblocking Destination Address, and provides the base address of the DMV buffer for reference frames 2 to 31. They are needed if the current B-Picture has MBs coded in direct mode. This is a private buffer used by the MPR hardware only. Its content is not accessed by software. All these buffers must be 64-byte cacheline aligned. There are a total of 32 possible Direct MV Read Buffers (not including the current write buffer of the current

MFX_AVC_DIRECTMODE_STATE

		picture) to read in the corresponding DMV. Each read buffer size is 557,056 bytes for 1 frame (the selected colPic). Scalable with frame height, but do not scale with frame width as the hardware assumes frame width (in MBs) fixed at 128 (smallest power of 2 value larger than 120 - 1920x1088 screen resolution). The adjacent DMV buffers are paired ([2 and 3], [4 and 5], [N and N+1], ..[30 and 31]). This field is changed to one per frame: both top and bottom field share the same Direct MV Buffer Base Address. **Programming Notes** This field is ignored if PreDeblockOutEnable is set to 0 (disable).
33	31:0	**Direct MV Buffer for Reference Frame 0 to 15 - Attributes** Format: MemoryAddressAttributes
34..35	63:0	**Direct MV Buffer for Write - Base Address** Format: SplitBaseAddress64ByteAligned This field provides the base address of the DMV write-only buffer for the current decoding frame/field. It is a private buffer used by the MPR hardware only. Its content is not accessed by software. All these buffers must be 64-byte cacheline aligned, i.e. the same as the above DMV read/write buffers. These 2 buffers can only be addressed by [img_dec_fs_idc[4:0]«1 + img_structure[1]] for the current picture being decoded. Each write buffer size is 557,056 bytes for 1 frame (the selected colPic). Scalable with frame height, but do not scale with frame width as the hardware assumes frame width (in MBs) fixed at 128 (smallest power of 2 value larger than 120 - 1920x1088 screen resolution). DMV write buffer 32 is valid only if the current picture is a progressive frame, MbAff frame, or a top field. DMV write buffer 33 is valid only if the current picture is a bottom field. GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and MBZ. Some GraphicsAddress fields only specify the upper address bits. For example, GraphicsAddress[47:12] is a 4KB page address.
36	31:0	**Direct MV Buffer for Write - Attributes** Format: MemoryAddressAttributes
37..70	1087:0	**POCList[34][31:0]** Each POC value is a signed 32-bit number. One-to-one correspondence with the 34 Direct MV Buffer Address for Reference and Currrent Frames/Fields There are 34 POC entries in the list. For reference picture, only the lower 32 POC [0-31] entries can be used, and POCList[ ] is indexed by the frame_store_ID[4:0], which is obtained from RefPicList L0/L1[RefPicIdx]. frame_Store_IDbit[0] (indicator for Top/Bottiom Field).For current picture, all 34 POC entries [0-33] can be addressed by POCList[img_dec_fs_idc[4:0]«1 + img_structure[1]].For frame-only mode, every other entry is skipped. For MBAFF and field-only picture, each entry is a field POC, and every two entries are paired.

MFX_AVC_REF_IDX_STATE

MFX_AVC_REF_IDX_STATE

Source:	VideoCS
Length Bias:	2

This is a slice level command and can be issued multiple times within a picture that is comprised of multiple slices. The same command is used for AVC encoder (PAK mode) and decoder (VLD mode); it is not need in decoder IT mode.

The inline data of this command is interpreted differently for encoder as for decoder. For decoder, it is interpreted as RefIdx List L0/L1 as in AVC spec., and it matches with the AVC API data structure for decoder in VLD mode : RefPicList[2][32] (L0:L1, 0:31 RefPic). But for encoder, it is interpreted as a Reference Index Mapping Table for L0 and L1 reference pictures. For packing the bits at the output of PAK, the syntax elements must follow the definition of RefIdxL0/L1 list according to the AVC spec. However, the decoder pipeline was designed to use a variation of that standard definition, as such a conversion (mapping) is needed to support the hardware design. The Reference lists are needed in processing both P and B slice in AVC codec. For P-MB, only L0 list is used; for B-MB both L0 and L1 lists are needed. For a B-MB that is coded in L1-only Prediction, only L1 list is used.

Programming Notes

An application will create the RefPicList L0 and L1 and pass onto the driver. The content of each entry of RefPicList L0/L1[ ] is a 7-bit picture index. This picture index is the same as that of RefFrameList[ ] content. This picture index, however, is not defined the same as the frame store ID (0 to 16, 5-bits) we have implemented in H/W. Hence, driver is required to manage a table to convert between picture index and intel frame store ID. As such, the final RefPicList L0/L1[ ] that the driver passes onto the H/W is not the same as that defined.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_AVC_REF_IDX_STATE Format: OpCode
	26:24	**Command Opcode** Default Value: 1h AVC Format: OpCode
	23:21	**SubOpcodeA** Default Value: 0h MFX_AVC_REF_IDX_STATE Format: OpCode
	20:16	**SubOpcodeB** Default Value: 4h MFX_AVC_REF_IDX_STATE Format: OpCode

<table>
<tr><th colspan="3">MFX_AVC_REF_IDX_STATE</th></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b><table><tr><td>Default Value:</td><td>0008h</td></tr><tr><td>Format:</td><td>=n</td></tr></table>Excludes DWords 0,1</td></tr>
<tr><td>1</td><td>31:1</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>RefPicList Select</b>
Num_ref_idx_l1_active is resulted from the specifications in both PPS and Slice Header for the current slice. However, since the full reference list L0 and/or L1 are always sent, only present flags are specified instead.
This parameter is specified for Intel interface only.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>RefPicList 0</td><td>The list that followed represents RefList L0 (Decoder VLD mode) or Ref Idx Mapping Table L0 (Encoder PAK mode)</td></tr><tr><td>1</td><td>RefPicList1</td><td>The list that followed represents RefList L1 (Decoder VLD mode) or Ref Idx Mapping Table L1 (Encoder PAK mode)</td></tr></table></td></tr>
<tr><td>2..9
<b>Programming Notes:</b>
HW supports only 1:1 reference index to reference picture mapping.
Reference index 0 should point to Reference picture 0, whose address is specified in MFX_PIPE_BUF_ADDR DW 19,20 (The reference picture numbers may be different from bit-stream reference picture)
Reference index1 should point to Reference picture2, whose address is specified in MFX_PIPE_BUF_ADDR 23, 24 (The reference picture numbers may be different from bit-stream reference picture)
Reference index2 should point to Reference picture4, whose address is specified in MFX_PIPE_BUF_ADDR 27,28 (The reference picture numbers</td><td>255:0</td><td><b>Reference List Entry</b>
This set of fields is always present whenever this command is issued.
It always specifies the full 32 reference pictures in the selected list, regardless they are "existing picture" or not. If a picture is non-existing, the corresponding entry should be set to all ones. Each list entry is 1 byte. A 32-bit DW can hold 4 list entries in the following format<ul><li>31:24 entry X+3 (e.g. listY_3)</li><li>23:16 entry X+2 (e.g. listY_2)</li><li>15:8 entry X+1 (e.g. listY_1)</li><li>7:0 entry X (e.g. listY_0)</li></ul>X is replaced by the paddr[2:0] * 4 ; paddr[5:0] with 0x20 and 0x27, and Y is replaced by 0 or 1.The byte definition for a reference picture :<ul><li>Bit 7 : Non-Existing - indicates that frame store index that should have been at this entry did not exist and was replaced by an index 0 (a valid entry) for error concealment</li></ul></td></tr>
</table>

MFX_AVC_REF_IDX_STATE		
may be different from bit-stream reference picture)		• Bit 6 : Long term bit - set this reference picture to be used as long term reference • Bit 5 : Field picture flag - indicates frame/field • Bit 4:0 : Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation) This is the final Reference List L0 or L1 after any reordering specified in the Slice Header as well as modified by the driver, and its indices values are all translated to the intel specification. If the reference picture is a frame (Bit5 = 1), frame store ID is always an even number. This list is used in outputting MV information by the BSD unit in VLD mode. DMV access also reads and writes Mvlist0 using this frame store ID. If this set of fields is interpreted as Reference Index Mapping Table L0/L1, the same field alignment is followed, i.e. 4 mapping entries per DW. Each mapping entry is one byte in size, but only the least significant 5 bits [4:0] is relevant. Driver should zero all the upper bits [7:5] for each entry.

MFX_AVC_SLICE_STATE

MFX_AVC_SLICE_STATE	
Source:	VideoCS
Length Bias:	2

This is a slice level command and can be issued multiple times within a picture that is comprised of multiple slices. The same command is used for AVC encoder (PAK mode) and decoder (VLD and IT modes).

Programming Notes
MFX_AVC_SLICE_STATE command is not issued for AVC Short Format Bitstream decode, instead MFD_AVC_SLICEADDR command is executed to retrieve the next slice MB Start Address X and Y by H/W itself.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_AVC_SLICE_STATE Format: OpCode
	26:24	**Command Opcode** Default Value: 1h AVC Format: OpCode
	23:21	**SubOpcodeA** Default Value: 0h MFX_AVC_SLICE_STATE Format: OpCode
	20:16	**Command SubOpcodeB** Default Value: 3h MFX_AVC_SLICE_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 8h DWORD_COUNT_n Format: =n Excludes DWords 0,1
1	31:4	**Reserved** Access: RO Format: MBZ
	3:0	**Slice Type** It is set to the value of the syntax element read from the Slice Header.

MFX_AVC_SLICE_STATE

Value	Name
0000b	P Slice
0001b	B Slice
0010b	I Slice
0011b-1111b	Reserved

Programming Notes
Bits[3:2] must be 0

2 | 31:30 | **Reserved**

Access:	RO
Format:	MBZ

29:24 | **Number of Reference Pictures in Inter-prediction List 1**

Format:	U6

This field is valid only for encoding a B Slice, for which it is expected to have at least one entry in the reference list L1; otherwise (if Slice Type is not a B Slice), this field must be set to 0.This field can be derived for a B Slice from the Slice Header syntax element NumRefIdxActiveMinus1 as, Num_Ref_Idx_L1 = NumRefIdxActiveMinus1[1] + 1.

Value	Name
0-32	

23:22 | **Reserved**

Access:	RO
Format:	MBZ

21:16 | **Number of Reference Pictures in Inter-prediction List 0**

Format:	U6

This field is valid for encoding a P or B Slice, for which it is expected to have at least one entry in the reference list L0; otherwise (if Slice Type is not a P or B Slice), this field must be set to 0.This field can be derived for a P or B Slice from the Slice Header syntax element NumRefIdxActiveMinus1 as, Num_Ref_Idx_L0 = NumRefIdxActiveMinus1[0] + 1.

Value	Name
0-32	

15:11 | **Reserved**

Access:	RO
Format:	MBZ

10:8 | **Log 2 Weight Denom Chroma**

Format:	U3

Value	Name
0-7	

<table>
<tr><th colspan="3">MFX_AVC_SLICE_STATE</th></tr>
<tr><td></td><td>7:3</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>2:0</td><td><b>Log 2 Weight Denom Luma</b>
<table>
<tr><td>Format:</td><td>U3</td></tr>
</table>
<table>
<tr><td>It is the base 2 logarithm of the denominator for all Luma weighting factors. It is set to the value of the syntax element read from the Slice Header Pred_Weight_Table().</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0-7</td><td></td></tr>
</table></td></tr>
<tr><td>3</td><td>31:30</td><td><b>Weighted Prediction Indicator</b>
This field indicates the Weighted Prediction mode for a P or B Slice. It is a combined field corresponding to the syntax element WeightedBiPredIdc or WeightedPredFlag read from the current active PPS.
• If it is a B-Slice, these bits are interpreted as:
00b - Specifies the default weighted inter-prediction to be applied01b - Specifies the explicit weighted inter-prediction to be applied10b - Specifies the implicit weighted inter-prediction to be applied11b - Reserved (not allowed)
• If it is a P Slice, these bits are interpreted as:
00b - Disables weighted inter-prediction (Default weighted)01b - Enables weighted inter-prediction (Explicit weighted)10b - 11b - Reserved
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Only when in B Slice with Weighted_Pred_Idc = 1 (explicit weighted prediction), will there be a L1 and/or a L0 weight+offset tables being sent to the BSD unit through the Slice_State command. Only when in P Slice with Weighted_Pred_Idc = 1, will there be a L0 weight+offset table being sent to the BSD.</td></tr>
<tr><td>If Weighted_Pred_Idc != 1 for B Slice or Weighted_Pred_Idc =0 for P Slice, no Slice_State command should be issued to send these tables. If still being issued, the data is read but ignored.</td></tr>
<tr><td>DXVA specifies Weighted_Bipred and Weighted_Pred in frame-level state. However, these two flags are combined and specified in slice level for both P and B slice type.</td></tr>
</table></td></tr>
<tr><td></td><td>29</td><td><b>Direct Prediction Type</b>
Type of direct prediction used for B Slices. This field is valid only for Slice_Type = B Slice; otherwise, it must be set to 0.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Temporal</td></tr>
<tr><td>1</td><td>Spatial</td></tr>
</table></td></tr>
</table>

MFX_AVC_SLICE_STATE

28:27 **Disable Deblocking Filter Indicator**

Value	Name	Description
00b		FilterInternalEdgesFlag is set equal to 1
01b		Disable all deblocking operation, no deblocking parameter syntax element is read; filterInternalEdgesFlag is set equal to 0
10b		Macroblocks in different slices are considered not available; filterInternalEdgesFlag is set equal to 1
11b	Reserved	Not defined in AVC

26 **Reserved**

Access:	RO
Format:	MBZ

25:24 **Cabac Init Idc[1:0]**

Specifies the index for determining the initialization table used in the context variable initialization process.

Value	Name
0-2	

Programming Notes
Cabac initialization is also dependent on the field/frame picture type, Slice type, and the current SliceQP value.

23:22 **Reserved**

Access:	RO
Format:	MBZ

21:16 **Slice Quantization Parameter**

Quantization Parameter for current slice. Derived from PPS and slice_delta_qp syntax element in Slice Header. It is needed for CABAC context initialization and deblocking filter control. And it is also used as the starting QP value in the very first MB of a slice. It is in the range of unsigned integer 0 to 51, for 8-bit pixel bit-depth.

15:12 **Reserved**

Access:	RO
Format:	MBZ

11:8 **Slice Beta Offset Div2**

Format:	S3

Range: [-6, 6] Inclusive
Specifies the offset used in accessing the deblocking filter strength tables.

7:4 **Reserved**

Access:	RO
Format:	MBZ

MFX_AVC_SLICE_STATE

	3:0	**Slice Alpha C0 Offset Div2** Format: S3 Range: [-6, 6] Inclusive Specifies the offset used in accessing the deblocking filter strength tables.
4	31:24	**Slice Vertical Position** This field specifies the position in y-direction of the first macroblock in the Slice in unit of macroblocks. The fields (Slice_MB_Start_Hor_Pos, Slice_MB_Start_Vert_Pos) are valid in VLD (decoding) mode only. They are ignored by hardware in decoding IT mode and encoding mode (whereas the position is provided by the per-macroblock object command).Derived **Programming Notes** Error Handling: Driver needs to check if FirstMbY starts at 0 on the first slice of frame. If not, driver needs to add a phantom slice with FirstMbX and FirstMbY set to 0.
	23:16	**Slice Horizontal Position** This field specifies the position in x-direction of the first macroblock in the Slice in unit of macroblocks. Derived **Programming Notes** Error Handling: Driver needs to check if FirstMbY starts at 0 on the first slice of frame. If not, driver needs to add a phantom slice with FirstMbX and FirstMbY set to 0.
	15	**Reserved** Access: RO Format: MBZ
	14:0	**Slice Start Mb Num** Exists If: //Decoder Only The MB number (linear MB address in a picture) at the start of a Slice, it must match with the Slice Horizontal Position (Slice_MB_Start_Hor_Pos) and Vertical Position (Slice_MB_Start_Vert_Pos) in the picture. **Programming Notes** In creating the Phantom Slice for error concealment, this field should set to the total number of MB in the current picture + 1.
5	31:24	**Reserved** Access: RO Format: MBZ
	23:16	**Next Slice Vertical Position** This field specifies the position in y-direction of the first macroblock in the next Slice in unit of macroblocks. This field is primarily used for error concealment. In the case that current slice is the last slice, this field should set to the height of picture (since y-direction is zero-based numbering).

<table>
<tr><th colspan="3">MFX_AVC_SLICE_STATE</th></tr>
<tr><td rowspan="2"></td><td>15:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>7:0</td><td><b>Next Slice Horizontal Position</b>
This field specifies the position in x-direction of the first macroblock in the next Slice in unit of macroblocks. This field is primarily used for error concealment. In the case that current slice is the last slice, this field should set to 0.</td></tr>
<tr><td rowspan="5">6
Encoder Only</td><td>31</td><td><b>Rate Control Counter Enable</b>
To enable the accumulation of bit allocation for rate control This field enables hardware Rate Control logic. The rest of the RC control fields are only valid when this field is set to 1. Otherwise, hardware ignores these fields.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Disable</td></tr><tr><td>1</td><td>Enable</td></tr></table></td></tr>
<tr><td>30</td><td><b>ResetRateControlCounter</b>
To reset the bit allocation accumulation counter to 0 to restart the rate control.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Not Reset</td></tr><tr><td>1</td><td>Reset</td></tr></table></td></tr>
<tr><td>29:28</td><td><b>RC Triggle Mode</b><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>00b</td><td>Always Rate Control</td><td>Whereas RC becomes active if sum_act > sum_target or sum_act < sum_target</td></tr><tr><td>01b</td><td>Gentle Rate Control</td><td>whereas RC becomes active if sum_act > upper_midpt or sum_act < lower_midpt</td></tr><tr><td>10b</td><td>Loose Rate Control</td><td>whereas RC becomes active if sum_act > sum_max or sum_act < sum_min</td></tr><tr><td>11b</td><td>Reserved</td><td></td></tr></table></td></tr>
<tr><td>27:24</td><td><b>RC Stable Tolerance</b><table><tr><td>Format:</td><td>U4</td></tr></table>This field specifies the tolerance required to deactivate RC once it has been triggered.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0-15</td><td></td></tr></table></td></tr>
<tr><td>23</td><td><b>RC Panic Enable</b>
If this field is set to 1, RC enters panic mode when sum_act > sum_max. RC Panic Type field controls what type of panic behavior is invoked.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Disable</td></tr><tr><td>1</td><td>Enable</td></tr></table></td></tr>
</table>

MFX_AVC_SLICE_STATE

22 **RC Panic Type**

This field selects between two RC Panic methods

Value	Name
0	QP Panic
1	CBP Panic

Programming Notes
If it is set to 0, in panic mode, the macroblock QP is maxed out, setting to requested QP + QP_max_pos_mod. If it is set to 1, for an intra macroblock, AC CBPs are set to zero (note that DC CBPs are not modified). For inter macroblocks, AC and DC CBPs are forced to zero.

21 **MB Type Direct Conversion Disable**

Exists If:	//B-Slice

For all Macroblock type conversions in different slices, refer to Section "Macroblock Type Conversion Rules" in the same volume.

Value	Name
0	Enable direct mode conversion
1	Disable direct mode conversion

Programming Notes
This field is zero for all other slices other than B-Slice.

20 **MB Type Skip Conversion Disable**

Exists If:	//P-Slice or B-Slice

For all Macroblock type conversions in different slices, refer to Section "Macroblock Type Conversion Rules" in the same volume.

Value	Name
0	Enable skip type conversion
1	Disable skip type conversion

Programming Notes
This field is zero for all other slices other than P_Slice or B-Slice. \

19 **Is Last Slice**

It is used by the zero filling in the Minimum Frame Size test.
Set this only for the last slice group

Value	Name	Description
1		Current slice is the last slice of a picture
0		Current slice is NOT the last slice of a picture

18 **Reserved**

<table>
<tr><th colspan="3">MFX_AVC_SLICE_STATE</th></tr>
<tr><td></td><td>17</td><td><b>Header Insertion Present in Bitstream</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td></td><td>No header insertion into the output bitstream buffer, in front of the current slice encoded bits.</td></tr>
<tr><td>1</td><td></td><td>Header insertion into the output bitstream buffer is present, and is in front of the current slice encoded bits.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This need to be set only for super slice0.</td></tr>
</table></td></tr>
<tr><td></td><td>16</td><td><b>SliceData Insertion Present in Bitstream</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td></td><td>No Slice Data insertion into the output bitstream buffer</td></tr>
<tr><td>1</td><td></td><td>Slice Data insertion into the output bitstream buffer is present.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This bit should be set for all super-slices.</td></tr>
</table></td></tr>
<tr><td></td><td>15</td><td><b>Tail Insertion Present in bitstream</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td></td><td>No tail insertion into the output bitstream buffer, after the current slice encoded bits</td></tr>
<tr><td>1</td><td></td><td>Tail insertion into the output bitstream buffer is present, and is after the current slice encoded bits.</td></tr>
</table></td></tr>
<tr><td></td><td>14</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>13</td><td><b>EmulationByteSliceInsertEnable</b>
To have PAK outputting SODB or EBSP to the output bitstream buffer
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td></td><td>outputting RBSP</td></tr>
<tr><td>1</td><td></td><td>outputting EBSP</td></tr>
</table></td></tr>
<tr><td></td><td>12</td><td><b>CabacZeroWordInsertionEnable</b>
To pad the end of a SliceLayer RBSP to meet the encoded size requirement.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td></td><td>No Cabac_Zero_Word Insertion</td></tr>
<tr><td>1</td><td></td><td>Allow internal Cabac_Zero_Word generation and append to the end of RBSP(effectively can be used as an indicator for last slice of a picture, if the assumption is only the last slice of a picture needs to insert CABAC_ZERO_WORDs.</td></tr>
</table></td></tr>
</table>

MFX_AVC_SLICE_STATE

	11:8	**Reserved** Access: | RO Format: | MBZ
	7:4	**Slice ID [3:0]** To identify the output data (coding information record) returned for rate control from PAK to ENC and VPP.
	3:2	**Reserved** Access: | RO Format: | MBZ
	1:0	**Stream ID [1:0]** To identify the output data (coding information record) returned for rate control from PAK to ENC and VPP.
7 Encoder Only	31:29	**Reserved** Access: | RO Format: | MBZ
	28:0	**Indirect PAK-BSE Data Start Address (Write)** Exists If: | //AVC Encode Mode This field specifies the memory starting address (offset) to write out the compressed bitstream data from the BSE processing. This pointer is relative to the MFC Indirect PAK-BSE Object Base Address. It is a byte-aligned address for the AVC bitstream data in both CABAC/CAVLC Modes. For Write, there is no need to have a data length field. It is assumed the global memory bound check specified in the IND_OBJ_BASE_ADDRESS command (Indirect PAK-BSE Object Access Upper Bound) will take care of any illegal write access. **Value** | **Name** [0h,1FFFFFFFh] |
8 Encoder Only	31:24	**Magnitude of QP Max Negative Modifier** Format: | U8 This field specifies the lower limit of the QP modifier. **Value** | **Name** 0-51 |
	23:16	**Magnitude of QP Max Positive Modifier** Format: | U8 This field specifies the upper limit of the QP modifier. **Value** | **Name** 0 - 15 |
	15:12	**Shrink Param - Shrink Resistance** Format: | U4 This field specifies the additional points added each time decreased correction is invoked.

<table>
<tr><th colspan="3">MFX_AVC_SLICE_STATE</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>0 - 15</td><td></td></tr></table></td></tr>
<tr><td></td><td>11:8</td><td><b>Shrink Param - Shrink Init</b><table><tr><td>Format:</td><td>U4</td></tr></table>This field specifies the initial points required to trip decreased control.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0 - 15</td><td></td></tr></table></td></tr>
<tr><td></td><td>7:4</td><td><b>Grow Param - Grow Resistance</b><table><tr><td>Format:</td><td>U4</td></tr></table>This field specifies the additional points added each time increased correction is invoked.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0 - 15</td><td></td></tr></table></td></tr>
<tr><td></td><td>3:0</td><td><b>Grow Param - Grow Init</b><table><tr><td>Format:</td><td>U4</td></tr></table>This field specifies the initial points required to trip increased control.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0 - 15</td><td></td></tr></table></td></tr>
<tr><td>9
Encoder Only</td><td>31</td><td><b>RoundInterEnable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>When this bit is not set, RoundInter defaults to 2.</td></tr>
<tr><td></td><td>30:28</td><td><b>RoundInter</b><table><tr><td>Format:</td><td>U3</td></tr></table>Rounding precision for Inter quantized coefficients<table><tr><th>Value</th><th>Name</th></tr><tr><td>000b</td><td>+1/16 <b>[Default]</b></td></tr><tr><td>001b</td><td>+2/16</td></tr><tr><td>010b</td><td>+3/16</td></tr><tr><td>011b</td><td>+4/16</td></tr><tr><td>100b</td><td>+5/16</td></tr><tr><td>101b</td><td>+6/16</td></tr><tr><td>110b</td><td>+7/16</td></tr><tr><td>111b</td><td>+8/16</td></tr></table></td></tr>
<tr><td></td><td>27</td><td><b>RoundIntraEnable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>When this bit is not set, RoundIntra defaults to 4.</td></tr>
<tr><td></td><td>26:24</td><td><b>RoundIntra</b><table><tr><td>Format:</td><td>U3</td></tr></table></td></tr>
</table>

MFX_AVC_SLICE_STATE

Rounding precision for Intra quantized coefficients

Value	Name
000b	+1/16 **[Default]**
001b	+2/16
010b	+3/16
011b	+4/16
100b	+5/16
101b	+6/16
110b	+7/16
111b	+8/16

23:20 **Correct 6**

Format:	U4

This field specifies the points used in the lowermost RC region when sum_act <= sum_min.

Value	Name
0 - 15	

19:16 **Correct 5**

Format:	U4

This field specifies the points used in the fifth RC region when sum_act > sum_min but <= lower_midpt.

Value	Name
0 - 15	

15:12 **Correct 4**

Format:	U4

This field specifies the points used in the fourth RC region when sum_act > lower_midpt but <= sum_target.

Value	Name
0 - 15	

11:8 **Correct 3**

Format:	U4

This field specifies the points used in the third RC region when sum_act > sum_target but <= upper_midpt.

Value	Name
0 - 15	

7:4 **Correct 2**

Format:	U4

This field specifies the points used in the second RC region when sum_act > upper_midpt but <= sum_max.

MFX_AVC_SLICE_STATE

Value	Name
0 - 15	

3:0 **Correct 1**

Format:	U4

This field specifies the points used in the topmost RC region when sum_act > sum_max.

Value	Name
0 - 15	

DWord	Bit	Description
10 Encoder Only	31:28	**ClampValues - CV7**
	27:24	**CV6**
	23:20	**CV5**
	19:16	**CV4**
	15:12	**CV3**
	11:8	**CV2**
	7:4	**CV1**
	3:0	**CV0 - Clamp Value 0**

Format:	U4

If the magnitude of coefficients at locations assigned with CV0 (mapping shown below) exceeds 2CV0-1, they are replaced with 2CV0-1. For coefficients at locations marked as 'none', no clamping is performed. The following mappings are only applied to luma and chroma blocks\subblocks containing AC coefficiencts (blocks\sublocks with only DC coeffs will not be clamped).

For 4x4 frame block, each coefficient is mapped to one of the eight CV values as following:

none	CV7	CV5	CV4
CV7	CV6	CV4	CV3
CV5	CV4	CV2	CV1
CV4	CV3	CV1	CV0

For 8x8 frame block, each coefficient is mapped to one of the eight CV values as following:

none	none	CV7	CV6	CV5	CV4	CV3	CV3
none	CV7	CV6	CV5	CV4	CV3	CV3	CV2
CV7	CV6	CV5	CV4	CV3	CV3	CV2	CV2
CV6	CV5	CV4	CV3	CV3	CV2	CV2	CV1
CV5	CV4	CV3	CV3	CV2	CV2	CV1	CV1
CV4	CV3	CV3	CV2	CV2	CV1	CV1	CV0
CV3	CV3	CV2	CV2	CV1	CV1	CV0	CV0
CV3	CV2	CV2	CV1	CV1	CV0	CV0	CV0

For 4x4 field block, each coefficient is mapped to one of the eight CV values as following:

MFX_AVC_SLICE_STATE

none	CV6	CV3	CV1
CV7	CV6	CV3	CV1
CV5	CV4	CV2	CV0
CV5	CV4	CV2	CV0

For 8x8 field block, each coefficient is mapped to one of the eight CV values as following:

none	none	CV6	CV5	CV4	CV3	CV2	CV1
none	CV7	CV6	CV5	CV4	CV3	CV2	CV1
CV7	CV6	CV5	CV4	CV3	CV3	CV2	CV1
CV7	CV6	CV5	CV4	CV3	CV2	CV2	CV1
CV6	CV5	CV4	CV4	CV3	CV2	CV1	CV0
CV6	CV5	CV4	CV3	CV2	CV2	CV1	CV0
CV5	CV5	CV4	CV3	CV2	CV1	CV1	CV0
CV5	CV5	CV4	CV3	CV2	CV1	CV1	CV0

Value	Name
0 - 15	

MFX_AVC_WEIGHTOFFSET_STATE

MFX_AVC_WEIGHTOFFSET_STATE

Source:	VideoCS
Length Bias:	2

This is a slice level command and can be issued multiple times within a picture that is comprised of multiple slices. The same command is used for AVC encoder (PAK mode) and decoder (VLD and IT modes). However, since for AVC decoder VLD and IT modes, and AVC encoder mode, the implicit weights are computed in hardware, this command is not issued. For encoder, regardless of the type of weight calculation is active for the current slice (default, implicit or explicit), they are all sent to the PAK as if they were all in explicit mode. However, for implicit weight and offset, each entry contains only a 16-bit weight and no offset (offset = 0 always in implicit mode and can be hard-coded inside the hardware). The weights (and offsets) are needed in processing both P and B slice in AVC codec. For P-MB, at most only L0 list is used; for B-MB both L0 and L1 lists may be needed. For a B-MB that is coded in L1-only Prediction, only L1 list is sent. The content of this command matches with the AVC API data structure for explicit prediction mode only : Weights[2][32][3][2] (L0:L1, 0:31 RefPic, Y:Cb:Cr, W:0)

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_ AVC_ WEIGHTOFFSET_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 5h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 60h Excludes DWord (0,1) Format: =n

<table>
<tr><th colspan="3">MFX_AVC_WEIGHTOFFSET_STATE</th></tr>
<tr><td>1</td><td>31:1</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>Weight and Offset Select</b>
It must be set in consistent with the WeightedPredFlag and WeightedBiPredIdc in the Img_State command. This parameter is specified for Intel interface only. For implicit even though only one entry may be used, still loading the whole 32-entry table.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Weight and Offset L0 table</td><td>The list that followed is associated with the weight and offset for RefPicList L0</td></tr><tr><td>1</td><td>Weight and Offset L1 table</td><td>The list that followed is associated with the weight and offset for RefPicList L1</td></tr></table></td></tr>
<tr><td>2..97</td><td>3071:0</td><td><b>WeightOffset</b><table><tr><td>WeightOffset[L=L0=0 or L1=1][i=0 to 31][Y=0/Cb=1/Cr=2][weight=0/offset=1]WeightOffset[L][i=0][Y=0][Weight=0], WeightOffset[L][i=0][Y=0][Offset=1]WeightOffset[L][i=0][Cb=1][Weight=0], WeightOffset[L][i=0][Cb=1][Offset=1]WeightOffset[L][i=0][Cr=2][Weight=0], WeightOffset[L][i=0][Cr=2][Offset=1]:WeightOffset[L][i=31][Y=0][Weight=0], WeightOffset[L][i=31][Y=0][Offset=1]WeightOffset[L][i=31][Cb=1][Weight=0], WeightOffset[L][i=31][Cb=1][Offset=1]WeightOffset[L][i=31][Cr=2][Weight=0], WeightOffset[L][i=31][Cr=2][Offset=1]</td></tr><tr><td>Format for explicit: Both Weight and Offset are S15 in two's compliment, with a valid range from -128 to 128Format for implicit: S15</td></tr><tr><td>This set of fields is always present whenever this command is issued. The full table, one entry for each reference picture, is always specified. Any reference list L0/L1[i] that does not exist, the corresponding weight and offset are set to 0.Weight and Offset are 2 byte each. A pair of Weight and Offset forms a dword, with Weight in the LOWER word and Offset in the HIGHER word. WeightOffset[L0=0][i=0 to 31][Y=0] (i.e. luma_weight_l0[i]) are specified for the weighting and offset factors applied to the luma prediction value for list 0 prediction using RefPicList0[i] (one-to-one correspondence in i). When luma_weight_l0_flag (Slice Header syntax element) is equal to 1, the value of luma_weight_l0[i] shall be in the range of -128 to 127. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to 2luma_log2_weight_denom for RefPicList0[i]. luma_log2_weight_denom is a Slice Header syntax element.WeightOffset[L0=0][i=0 to 31][Cb=1] (i.e. chromaCb_weight_l0[i]) are specified for the weighting and offset factors applied to the chroma Cb prediction values for list 0 prediction using RefPicList0[i] (one-to-one correspondence in i). When chroma_weight_l0_flag (Slice Header syntax element) is equal to 1, the value of chromaCb_weight_l0[i] shall be in the range of -128 to 127. When chroma_weight_l0_flag is equal to 0, chromaCb_weight_l0[i] shall be inferred to be equal to 2chroma_log2_weight_denom for RefPicList0[i]. chroma_log2_weight_denom is a Slice Header syntax element.WeightOffset[L0=0][i=0 to 31][Cr=2] (i.e. chromaCr_weight_l0[i]) are specified for the weighting and offset factors applied to the chroma Cr prediction values for list 0 prediction using RefPicList0[i] (one-to-one correspondence in i). When chroma_weight_l0_flag (Slice Header syntax element) is equal to 1, the value of</td></tr></table></td></tr>
</table>

MFX_AVC_WEIGHTOFFSET_STATE		
		chromaCr_weight_l0[i] shall be in the range of -128 to 127. When chroma_weight_l0_flag is equal to 0, chromaCr_weight_l0[i] shall be inferred to be equal to 2chroma_log2_weight_denom for RefPicList0[i].

MFX_BSP_BUF_BASE_ADDR_STATE

MFX_BSP_BUF_BASE_ADDR_STATE

Source:	VideoCS
Length Bias:	2

This frame-level state command is used to specify all the buffer base addresses needed for the operation of the AVC Bit Stream Processing Units (for decoder, it is BSD Unit; for encoder, it is BSE Unit)For both encoder and decoder, currently it is assumed that all codec standards can share the same BSP_BUF_BASE_STATE. The simplicity of this command is the result of moving all the direct MV related processing into the ENC Subsystem. Since all implicit weight calculations and direct MV calculations are done in ENC and all picture buffer management are done in the Host, there is no need to provide POC (POC List - FieldOrderCntList, CurrPic POC - CurrFieldOrderCnt) information to PAK. For decoder, all the direct mode information are sent in a separate slice-level command (AVC_DIRECTMODE_STATE command).In addition, in Encoder, the row stores for CABAC encoding and MB Parameters Construction (MPC) are combined into one single row store. The row stores specified in this command do not combine with those specified in the MFC_PIPE_BUF_ADDR_STATE command for hardware simplification reason.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Pipeline Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 4h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 8h Excludes DWord (0,1) Format: =n

<table>
<tr><th colspan="3">MFX_BSP_BUF_BASE_ADDR_STATE</th></tr>
<tr><td>1</td><td>31:6</td><td><b>BSD/MPC Row Store Scratch Buffer Base Address - Read/Write</b>
<table>
<tr><td>This field provides the base address of the scratch buffer used by BSD (decoder) and MPC (encoder) unit to store MB information of the previous row for coding each macroblockin the current row. It is a private buffer used by the BSD (decoder) and MPC (encoder)hardware only. Its content is not accessible by software. This Row Store buffer must be64-byte cacheline aligned. Hardware uses the horizontal address of the current macroblock to address this Row Store.</td></tr>
<tr><td>For AVC BSD, 2 cacheline (CL) per MB when in MBAFF mode (row of MB pair); 1 CL per MB for non-MBAFF. So, to support 256 MBs per row (4K screen resolution), 2 * 256 * 64 bytes= 32,768 bytes are required. Cacheline alignment should be followed. For AVC MPC, 1cachline for non-MBAFF, 2 cachelines for MBAFF per MB. For VC1, the BSD row store is512-bit (one cacheline) per MB, times the number of MBs per picture MB row.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This is one of the four RowStore Scratch Buffers which can programmed to use the internal Media Storage (total size 640 CacheLine). When Deblocking Filter Row Store Scratch Buffer Cache Select is programmed to "1", this will be stored inside MFX Media Internal Storage. Driver then needs to program this Base Address between 0 to 639, indicating starting cacheline address location for this buffer. Driver needs to make sure the whole buffer fits into Media Internal Storage.
<i>(Notes: 1 cachelines per MB for non-mbaff; 2 cachelines per MB pair for mbaff, and the buffer needs to have enough space for 1 MB (pair) row).</i></td></tr>
</table></td></tr>
<tr><td></td><td>5:0</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>2</td><td>31:16</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>15:0</td><td><b>BSD/MPC Row Store Scratch Buffer Base Address - Read/Write [47:32]</b>
<table>
<tr><td>This field is for the upper range of BSD/MPC Row Store Scratch Buffer Base Address.</td></tr>
<tr><td>This field is used for 48-bit addressing.</td></tr>
</table></td></tr>
<tr><td>3</td><td>31:15</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>14:13</td><td><b>BSD/MPC Row Store Scratch Buffer - Tiled Resource Mode</b>
<table>
<tr><td>Format:</td><td>U2</td></tr>
</table>
<b>For Media Surfaces:</b> This field specifies the tiled resource mode.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>TRMODE_NONE</td><td>No tiled resource</td></tr>
<tr><td>1h</td><td>TRMODE_TILEYF</td><td>4KB tiled resources</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">MFX_BSP_BUF_BASE_ADDR_STATE</th></tr>
<tr><td></td><td></td><td><table><tr><td>2h</td><td>TRMODE_TILEYS</td><td>64KB tiled resources</td></tr><tr><td>3h</td><td>Reserved</td><td></td></tr></table></td></tr>
<tr><td></td><td>12</td><td><b>BSD/MPC Row Store Scratch Buffer Cache Select</b>
This field controls if Intra Row Store is going to store inside Media Internal Storage or to LLC.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td></td><td>Buffer going to LLC</td></tr><tr><td>1</td><td></td><td>Buffer going to Internal Media Storage</td></tr></table></td></tr>
<tr><td></td><td>11:9</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>BSD/MPC Row Store Scratch Buffer - Arbitration Priority Control</b><table><tr><td>Format:</td><td>U2</td></tr></table>This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>Highest priority</td></tr><tr><td>01b</td><td>Second highest priority</td></tr><tr><td>10b</td><td>Third highest priority</td></tr><tr><td>11b</td><td>Lowest priority</td></tr></table></td></tr>
<tr><td></td><td>6:1</td><td><b>BSD/MPC Row Store Scratch Buffer - Index to Memory Object Control State (MOCS) Tables</b><table><tr><td>Format:</td><td>U6</td></tr></table>The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.</td></tr>
<tr><td></td><td>0</td><td><b>Reserved</b></td></tr>
<tr><td>4</td><td>31:6</td><td><b>MPR Row Store Scratch Buffer Base Address - Read/Write (Decoder Only)</b>
This field provides the base address of the scratch buffer used by decoder's MPR unit to store MB information of the previous row for decoding each macroblock in the current row. It is a private buffer used by the MPR hardware only. Its content is not accessible by software.<table><tr><th>Programming Notes</th></tr><tr><td>The MPR Row Store buffer must be 64-byte cacheline aligned. Hardware uses the horizontal address of each macroblock to address the MPR Row Store. Except ILDB Control Data, all other operations do not cross slice boundary. This field is specified in frame-level.2 cacheline (CL) per MB when in MBAFF mode (row of MB pair); 1 CL per MB for non-MBAFF, So, to support 256 MBs per row (4K screen resolution), 2 * 256 * 64 bytes = 32,768 bytes are required. Cacheline alignment should be followed. This field is only valid for AVC decoder mode</td></tr><tr><td>This is one of the four RowStore Scratch Buffers which can programmed to use the internal Media Storage (total size 640 CacheLine). When Deblocking Filter Row Store Scratch Buffer Cache Select is programmed to "1", this will be cache inside MFX Media Internal Storage. Driver then needs to program this Base Address between 0 to 639, indicating starting cachelines</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFX_BSP_BUF_BASE_ADDR_STATE</th></tr>
<tr><td></td><td></td><td>address location for this buffer. Driver needs to make sure the whole buffer fits into Media Internal Storage
(Notes: 1 cachelines per MB for non-mbaff; 2 cachelines per MB pair for mbaff, and the buffer needs to have enough space for 1 MB (pair) row).</td></tr>
<tr><td></td><td>5:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>5</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>MPR Row Store Scratch Buffer Base Address - Read/Write [47:32]</b>
This field is for the upper range of MPR Row Store Scratch Buffer Base Address.This field is used for 48-bit addressing.</td></tr>
<tr><td>6</td><td>31:15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:13</td><td><b>MPR Row Store Scratch Buffer - Tiled Resource Mode</b><table><tr><td>Format:</td><td>U2</td></tr></table><b>For Media Surfaces:</b>This field specifies the tiled resource mode.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>TRMODE_NONE</td><td>No tiled resource</td></tr><tr><td>1h</td><td>TRMODE_TILEYF</td><td>4KB tiled resources</td></tr><tr><td>2h</td><td>TRMODE_TILEYS</td><td>64KB tiled resources</td></tr><tr><td>3h</td><td>Reserved</td><td></td></tr></table></td></tr>
<tr><td></td><td>12</td><td><b>MPR Row Store Scratch Buffer Cache Select</b>
This field controls if Intra Row Store is going to store inside Media Internal Storage or to LLC.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td></td><td>Buffer going to LLC</td></tr><tr><td>1</td><td></td><td>Buffer going to Internal Media Storage</td></tr></table></td></tr>
<tr><td></td><td>11:9</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>MPR Row Store Scratch Buffer - Arbitration Priority Control</b><table><tr><td>Format:</td><td>U2</td></tr></table>This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>Highest priority</td></tr><tr><td>01b</td><td>Second highest priority</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFX_BSP_BUF_BASE_ADDR_STATE</th></tr>
<tr><td></td><td></td><td><table><tr><td>10b</td><td>Third highest priority</td></tr><tr><td>11b</td><td>Lowest priority</td></tr></table></td></tr>
<tr><td></td><td>6:1</td><td><b>MPR Row Store Scratch Buffer - Index to Memory Object Control State (MOCS) Tables</b><table><tr><td>Format:</td><td>U6</td></tr></table>The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.</td></tr>
<tr><td></td><td>0</td><td><b>Reserved</b></td></tr>
<tr><td>7</td><td>31:6</td><td><b>Bitplane Read Buffer Base Address</b>
It must be cacheline aligned (i.e. 64 bytes address boundary), so lower bit 0 to 5 are used for controlling information.(In Cantiga, this field must be dword aligned.)Bitplane buffer is a linear buffer. In VC1 Long format, it is written by an application. In VC1 Short Format, it is written and read by H/W only. For VC1 intel Long Format : it is a read-only bufferFor VC1 DXVA2 Short Format : it is a write and a read buffer. This field is only valid for VC1 decoder mode.</td></tr>
<tr><td></td><td>5:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>8</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Bitplane Read Buffer Base Address - Read/Write [47:32]</b>
This field is for the upper range of Bitplane Read Buffer Base Address. This field is used for 48-bit addressing.</td></tr>
<tr><td>9</td><td>31:15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:13</td><td><b>Bitplane Read Buffer - Tiled Resource Mode</b><table><tr><td>Format:</td><td>U2</td></tr></table><b>For Media Surfaces:</b> This field specifies the tiled resource mode.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>TRMODE_NONE</td><td>No tiled resource</td></tr><tr><td>1h</td><td>TRMODE_TILEYF</td><td>4KB tiled resources</td></tr><tr><td>2h</td><td>TRMODE_TILEYS</td><td>64KB tiled resources</td></tr><tr><td>3h</td><td>Reserved</td><td></td></tr></table></td></tr>
<tr><td></td><td>12:9</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFX_BSP_BUF_BASE_ADDR_STATE</th></tr>
<tr><td></td><td>8:7</td><td><b>Bitplane Read Buffer - Arbitration Priority Control</b>
<table>
<tr><td>Format:</td><td>U2</td></tr>
</table>
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>Highest priority</td></tr>
<tr><td>01b</td><td>Second highest priority</td></tr>
<tr><td>10b</td><td>Third highest priority</td></tr>
<tr><td>11b</td><td>Lowest priority</td></tr>
</table>
</td></tr>
<tr><td></td><td>6:1</td><td><b>Bitplane Read Buffer - Index to Memory Object Control State (MOCS) Tables</b>
<table>
<tr><td>Format:</td><td>U6</td></tr>
</table>
The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
</td></tr>
<tr><td></td><td>0</td><td><b>Reserved</b></td></tr>
</table>

MFX_DBK_OBJECT

MFX_DBK_OBJECT		
Source:	VideoCS	
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_DBK_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h Common Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 0Bh Excludes DWord (0,1) Format: =n Note: Regardless of the mode, inline data must be present in this command
1	31:6	**Pre Deblocking Source Address** Format: GraphicsAddress[31:6] Specifies the 4K byte aligned frame buffer address for outputting the non-filtered reconstructed YUV picture (i.e. output of final adder in each codec standard, and prior to the deblocking filter unit).
	5:0	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">MFX_DBK_OBJECT</th></tr>
<tr><td>2</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Pre Deblocking Source Address High</b><table><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table>This field is for the upper range of Pre-Deblocking Source Address. This field is used for 48-bit addressing.</td></tr>
<tr><td>3</td><td>31:15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:13</td><td><b>Pre Deblocking Source - Tiled Resource Mode</b>
<b>For Media Surfaces:</b> This field specifies the tiled resource mode.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>TRMODE_NONE</td><td>No tiled resource</td></tr><tr><td>1h</td><td>TRMODE_TILEYF</td><td>4KB tiled resources</td></tr><tr><td>2h</td><td>TRMODE_TILEYS</td><td>64KB tiled resources</td></tr><tr><td>3h</td><td>Reserved</td><td></td></tr></table></td></tr>
<tr><td></td><td>12:11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Pre Deblocking Source - Memory Compression Mode</b>
Distinguishes Vertical from Horizontal compression. Please refer to <b>Memory Data Formats</b>, <b>Media Memory Compression</b> for more details.<table><tr><th>Value</th><th>Name</th></tr><tr><td>1</td><td>Vertical Compression Mode</td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>Pre Deblocking Source - Memory Compression Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>Memory compression will be attempted for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr><tr><td>1</td><td>Compression Enable</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Pre Deblocking Source - Arbitration Priority Control</b>
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>Highest priority</td></tr><tr><td>01b</td><td>Second highest priority</td></tr><tr><td>10b</td><td>Third highest priority</td></tr><tr><td>11b</td><td>Lowest priority</td></tr></table></td></tr>
</table>

MFX_DBK_OBJECT

	6:1	**Pre Deblocking Source - Index to Memory Object Control State (MOCS) Tables** The index to define the L3 and system cache memory properties. The details of the controls arefurther defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
	0	**Reserved**
4	31:6	**Deblocking Control Address** Format: GraphicsAddress[31:6] Specifies the 4K byte aligned frame buffer address as input MB-level deblocking parameters to control the way hardware deblock each micro-block. One 512-bit cacheline is allocated for each Macroblock in raster scan order.
	5:0	**Reserved** Access: RO Format: MBZ
5	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Deblocking Control Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Deblocking Control Address (DeblockCntrlAddr). This field is used for 48-bit addressing.
6	31:15	**Reserved** Access: RO Format: MBZ
	14:13	**Deblocking Control - Tiled Resource Mode** **For Media Surfaces:** This field specifies the tiled resource mode.
	12:11	**Reserved** Access: RO Format: MBZ
	10	**Deblocking Control - Memory Compression Mode** Distinguishes Vertical from Horizontal compression. Please refer to **Memory Data Formats**, **Media Memory Compression** for more details.

Deblocking Control - Tiled Resource Mode (DWord 6, bits 14:13):

Value	Name	Description
0h	TRMODE_NONE	No tiled resource
1h	TRMODE_TILEYF	4KB tiled resources
2h	TRMODE_TILEYS	64KB tiled resources
3h	Reserved	

<table>
<tr><th colspan="3">MFX_DBK_OBJECT</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Horizontal Compression Mode</td></tr><tr><td>1</td><td>Vertical Compression Mode</td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>Deblocking Control - Memory Compression Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>Memory compression will be attempted for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Deblocking Control - Arbitration Priority Control</b>
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>Highest priority</td></tr><tr><td>01b</td><td>Second highest priority</td></tr><tr><td>10b</td><td>Third highest priority</td></tr><tr><td>11b</td><td>Lowest priority</td></tr></table></td></tr>
<tr><td></td><td>6:1</td><td><b>Deblocking Source - Index to Memory Object Control State (MOCS) Tables</b>
The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.</td></tr>
<tr><td></td><td>0</td><td><b>Reserved</b></td></tr>
<tr><td>7</td><td>31:6</td><td><b>Deblocking Destination Address</b><table><tr><td>Format:</td><td>GraphicsAddress[31:6]</td></tr></table>Specifies the 4K byte aligned frame buffer address for outputting the post-loop filtered reconstructed YUV picture (i.e. output of the deblocking filter unit)</td></tr>
<tr><td></td><td>5:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>8</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Deblocking Destination Address High</b><table><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table>This field is for the upper range of Deblocking Destination Address. This field is used for 48-bit addressing.</td></tr>
<tr><td>9</td><td>31:15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFX_DBK_OBJECT</th></tr>
<tr><td></td><td>14:13</td><td><b>Deblocking Destination - Tiled Resource Mode</b>
<b>For Media Surfaces:</b> This field specifies the tiled resource mode.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>TRMODE_NONE</td><td>No tiled resource</td></tr>
<tr><td>1h</td><td>TRMODE_TILEYF</td><td>4KB tiled resources</td></tr>
<tr><td>2h</td><td>TRMODE_TILEYS</td><td>64KB tiled resources</td></tr>
<tr><td>3h</td><td>Reserved</td><td></td></tr>
</table></td></tr>
<tr><td></td><td>12:11</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>10</td><td><b>Deblocking Destination - Memory Compression Mode</b>
Distinguishes Vertical from Horizontal compression. Please refer to <b>Memory Data Formats</b>, <b>Media Memory Compression</b> for more details.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Horizontal Compression Mode</td></tr>
</table></td></tr>
<tr><td></td><td>9</td><td><b>Deblocking Destination - Memory Compression Enable</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
Memory compression will be attempted for this surface.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Compression Disable</td></tr>
</table></td></tr>
<tr><td></td><td>8:7</td><td><b>Deblocking Destination - Arbitration Priority Control</b>
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>Highest priority</td></tr>
<tr><td>01b</td><td>Second highest priority</td></tr>
<tr><td>10b</td><td>Third highest priority</td></tr>
<tr><td>11b</td><td>Lowest priority</td></tr>
</table></td></tr>
<tr><td></td><td>6:1</td><td><b>Deblocking Destination - Index to Memory Object Control State (MOCS) Tables</b>
The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.</td></tr>
<tr><td></td><td>0</td><td><b>Reserved</b></td></tr>
<tr><td>10</td><td>31:6</td><td><b>Deblock Row Store Address</b>
<table>
<tr><td>Format:</td><td>GraphicsAddress[31:6]</td></tr>
</table>
This field provides the base address of the scratch buffer (read and write) used by the deblocking filter unit to store MB information of the previous row for filtering of each macroblock in the current row. The Deblocking Filter Row Store buffer must be 64-byte cacheline aligned. Hardware uses the horizontal address of the current macroblock to address the</td></tr>
</table>

<table>
<tr><th colspan="3">MFX_DBK_OBJECT</th></tr>
<tr><td></td><td></td><td>Deblocking Filter Row Store.</td></tr>
<tr><td></td><td>5:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>11</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Deblock Row Store Address High</b><table><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table>This field is for the upper range of Deblock Row Store Address (DeblockRowStoreAddr). This field is used for 48-bit addressing.</td></tr>
<tr><td>12</td><td>31:15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:13</td><td><b>Deblock Row Store - Tiled Resource Mode</b>
<b>For Media Surfaces:</b> This field specifies the tiled resource mode.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>TRMODE_NONE</td><td>No tiled resource</td></tr><tr><td>1h</td><td>TRMODE_TILEYF</td><td>4KB tiled resources</td></tr><tr><td>2h</td><td>TRMODE_TILEYS</td><td>64KB tiled resources</td></tr><tr><td>3h</td><td>Reserved</td><td></td></tr></table></td></tr>
<tr><td></td><td>12:11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Deblock Row Store - Memory Compression Mode</b>
Distinguishes Vertical from Horizontal compression. Please refer to <b>Memory Data Formats</b>, <b>Media Memory Compression</b> for more details.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Horizontal Compression Mode</td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>Deblock Row Store - Memory Compression Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>Memory compression will be attempted for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFX_DBK_OBJECT</th></tr>
<tr><td></td><td>8:7</td><td><b>Deblock Row Store - Arbitration Priority Control</b>
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>Highest priority</td></tr>
<tr><td>01b</td><td>Second highest priority</td></tr>
<tr><td>10b</td><td>Third highest priority</td></tr>
<tr><td>11b</td><td>Lowest priority</td></tr>
</table></td></tr>
<tr><td></td><td>6:1</td><td><b>Coeff Probability StreamIn Address - Index to Memory Object Control State (MOCS) Tables</b>
The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.</td></tr>
<tr><td></td><td>0</td><td><b>Reserved</b></td></tr>
</table>

MFX_FQM_STATE

MFX_FQM_STATE

Source:	VideoCS
Length Bias:	2

This is a common state command for AVC encoder modes. For encoder, it represents both the forward QM matrices as well as the decoding QM matrices. This is a Frame-level state. Only Scaling Lists specified by an application are being sent to the hardware. The driver is responsible for determining the final set of scaling lists to be used for decoding the current slice, based on the AVC Spec Table 7-2 (Fall-Back Rules A and B). In MFX AVC PAK mode, PAK needs both forward Q scaling lists and IQ scaling lists. The IQ scaling lists are sent as in MFD in raster scan order. But the Forward Q scaling lists are sent in column-wise raster order (column-by-column) to simplify the H/W. Driver will perform all the scan order conversion for both ForwardQ and IQ.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 20h Excludes DWord (0,1) Format: =n
1	31:2	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">MFX_FQM_STATE</th></tr>
<tr><td></td><td>1:0</td><td><b>AVC</b>
<table><tr><td>Exists If:</td><td>//AVC- Decoder Only</td></tr></table>
<b>For AVC QM Type</b>: This field specifies which Quantizer Matrix is loaded.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>AVC_4x4_Intra_MATRIX, (Y-4DWs, Cb-4DWs, Cr-4DWs, reserved-4DWs)</td></tr>
<tr><td>1</td><td>AVC_4x4_Inter_MATRIX, (Y-4DWs, Cb-4DWs, Cr-4DWs, reserved-4DWs)</td></tr>
<tr><td>2</td><td>AVC_8x8_Intra_MATRIX</td></tr>
<tr><td>3</td><td>AVC_8x8_Inter_MATRIX</td></tr>
</table></td></tr>
<tr><td></td><td>1:0</td><td><b>MPEG2</b>
<table><tr><td>Exists If:</td><td>//MPEG2- Decoder Only</td></tr></table>
<b>For MPEG2 QM Type</b>: This field specifies which Quantizer Matrix is loaded.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>MPEG_INTRA_QUANTIZER_MATRIX</td></tr>
<tr><td>1</td><td>MPEG_NON_INTRA_QUANTIZER_MATRIX</td></tr>
<tr><td>2-3</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td></td><td>1:0</td><td><b>JPEG</b>
<table><tr><td>Exists If:</td><td>//JPEG- Encoder Only</td></tr></table>
<b>For JPEG QM Type</b>: This field specifies which Quantizer Matrix is loaded.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>JPEG_Luma_Y_QUANTIZER_MATRIX (or R)</td></tr>
<tr><td>1</td><td>JPEG_Chroma_Cb_QUANTIZER_MATRIX (or G)</td></tr>
<tr><td>2</td><td>JPEG_Chroma_Cr_QUANTIZER_MATRIX (or B)</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>For JPEG encoder, each quantization element presents 16-bit 1/QM[i][j]. In RGB encoding, because the order input image components can be RGB, GBR, BGR, YUV, the value 0 is used for the first image component, the value 1 is used for the second image component, and the value 2 is used for the third image component.</td></tr>
</table></td></tr>
<tr><td>2..33</td><td>1023:0</td><td><b>Forward Quantizer Matrix</b>
The format of a Quantizer Matrix is an 8x8 matrix in raster order. Each element is an unsigned byte.</td></tr>
</table>

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

Source:	VideoCS
Length Bias:	2

This state command provides the memory base addresses for all row stores, StreamOut buffer and reconstructed picture output buffers required by the MFD or MFC Engine (that are in addition to the row stores of the Bit Stream Decoding/Encoding Unit (BSD/BSE) and the reference picture buffers).This is a picture level state command and is common among all codec standards and for both encoder and decoder operating modes. However, some fields may only applicable to a specific codec standard. All Pixel Surfaces (original, reference frame and reconstructed frame) in the Encoder are programmed with the same surface state (NV12 and TileY format), except each has its own frame buffer base address. In the tile format, there is no need to provide buffer offset for each slice; since from each MB address, the hardware can calculate the corresponding memory location within the frame buffer directly.

The MFX_IND_OBJ_BASE_ADDR command sets the memory base address pointers for the corresponding Indirect Object Data Start Addresses (Offsets) specified in each OBJECT commands. The characteristic of these indirect object data is their variable size (per MB or per Slice).Hence, each OBJECT command must specify the indirect object data offset from the base address to start fetching or writing object data.

While the use of base address is unconditional, the indirection can be effectively disabled by setting the base address to zero.
For decoder, there are:

- 1 read-only per-slice indirect object in the BSD_OBJECT Command, and
- 2 read-only per-MB indirect objects in the IT_OBJECT Command.

For decoder: the Video Command Streamer (VCS) will perform the memory access bound check automatically using the corresponding MFC Indirect Object Access Upper Bound specification. If any access is at or beyond the upper bound, zero value is returned. The request to memory is still being sent, but the corresponding codec's BSD unit will detect this condition and perform the zeroing return. If the Upper Bound is turned off, the beyond bound request will return whatever on the bus (invalid data).
For encoder, there are:

- 1 read-only per-MB indirect object in the PAK_OBJECT Command, and
- 1 write-only per-slice indirect object in the PAK Slice_State Command

For encoder: whenever an out of bound address accessing request is generated, VMX will detect such requests and snap the address to the corresponding [indirect object base address + indirect data start address]. VMX will return all 0s as the data to the requestor. NotationDefinitionPhysicalAddress[n:m] Corresponding bits of a physical graphics memory byte address (not mapped by a GTT) GraphicsAddress[n:m] Corresponding bits of an absolute, virtual graphics memory byte address (mapped by a GTT).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode

MFX_IND_OBJ_BASE_ADDR_STATE

	28:27	**Pipeline** Default Value: 2h MFX_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	26:24	**Common Opcode** Default Value: 0h MFX_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	23:21	**Sub OpcodeA** Default Value: 0h MFX_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	20:16	**SubOpcodeB** Default Value: 3h MFX_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 0018h Excludes DWord (0,1) Format: =n
1..2	63:0	**MFX Indirect Bitstream Object - Base Address** Format: SplitBaseAddress4KByteAligned Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the MFD_XXX_BSD_OBJECT command for fetching (reading) the compressed Slice Data. This field is only valid in MPEG2, AVC and VC1 decoder VLD mode.
3	31:0	**MFX Indirect Bitstream Object - Attributes** Format: MemoryAddressAttributes
4..5	63:0	**MFX Indirect Bitstream Object - Upper Bound** Format: SplitBaseAddress4KByteAligned This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the MFD_XXX_BSD_OBJECT command for the compressed Slice Data. Indirect data accessed at this address and beyond will return as 0 by the hardware. Setting this field to 0 will cause this range check to be ignored. If non-zero, this address must be greater than the MFX Indirect Bitstream ObjectBase Address state. Hardware ignores this field if indirect data is not present, i.e. the Indirect Data Length field of the MFD_XXX_BSD_OBJECT command is set to 0.This field is only valid in MPEG2, AVC, VP8, and VC1 decoder VLD mode. **Programming Notes** For **VP8 Encoder**, this field is corresponding to **MFC Indirect PAK-BSE Object - Access Upper Bound in DW24, DW25**. Please program Indirect bitstream upper bound in this field the same as DW24, DW25.

MFX_IND_OBJ_BASE_ADDR_STATE		
6..7	63:0	**MFX Indirect MV Object - Base Address** Format: **SplitBaseAddress4KByteAligned** Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the encoder MFC_AVC_PAK_OBJECT command or the decoder MFD_IT_OBJECT command for fetching the per-MB MV data. This field is only valid in AVC encoder mode or in AVC decoder IT mode
8	31:0	**MFX Indirect MV Object - Attributes** Format: **MemoryAddressAttributes**
9..10	63:0	**MFX Indirect MV Object - Upper Bound** Format: **SplitBaseAddress4KByteAligned** This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the MFC_AVC_PAK_OBJECT / MFD_IT_OBJECT command for the per-MB MV data. Indirect data accessed at this address and beyond will return as 0 by the hardware. Setting this field to 0 will cause this range check to be ignored. If non-zero, this address must be greater than the MFX Indirect MV Object Base Address state. Hardware ignores this field if indirect data is not present, i.e. the Indirect Data Length field of the MFC_AVC_PAK_OBJECT / MFD_IT_OBJECT command is set to 0.This field is only valid in AVC encoder mode or in AVC decoder IT mode.
11..12	63:0	**MFD Indirect IT-COEFF Object - Base Address** Format: **SplitBaseAddress4KByteAligned** Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the MFD_IT_OBJECT command for fetching (reading) the per-MB non-scaled coefficient data (all inverse scaling and quantization are done in hardware). This field is only valid in MPEG2, AVC and VC1 decoder IT mode.
13	31:0	**MFD Indirect IT-COEFF Object - Attributes** Format: **MemoryAddressAttributes**
14..15	63:0	**MFD Indirect IT-COEFF Object - Upper Bound** Format: **SplitBaseAddress4KByteAligned** This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the MFD_IT_OBJECT command for the per-MB non-scaled coefficient data. Indirect data accessed at this address and beyond will return as 0 by the hardware. Setting this field to 0 will cause this range check to be ignored. If non-zero, this address must be greater than the MFD Indirect IT-COEFF Object Base Address state. Hardware ignores this field if indirect data is not present, i.e. the Indirect COEFF Data Length field of the MFD_IT_OBJECT command is set to 0.This field is only valid in MPEG2, AVC and VC1 decoder IT mode.
16..17	63:0	**MFD Indirect IT-DBLK Object - Base Address** Format: **SplitBaseAddress4KByteAligned** Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the MFD_IT_OBJECT command for fetching (reading) the per-MB Deblocking filter control data. This field is only valid in AVC decoder IT mode.
18	31:0	**MFD Indirect IT-DBLK Object - Attributes** Format: **MemoryAddressAttributes**

MFX_IND_OBJ_BASE_ADDR_STATE		
19..20	63:0	**MFD Indirect IT-DBLK Object - Upper Bound** Format: SplitBaseAddress4KByteAligned This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the MFD_IT_OBJECT command for the per-MB Deblocking filter control data. Indirect data accessed at this address and beyond will return as 0 by the hardware. Setting this field to 0 will cause this range check to be ignored. If non-zero, this address must be greater than the MFD Indirect IT-DBLK Object Base Address state. Hardware ignores this field if indirect data is not present, i.e. the Indirect Deblocking Control Data Length field of the MFD_IT_OBJECT command is set to 0.This field is only valid in AVC decoder IT mode.
21..22	63:0	**MFC Indirect PAK-BSE Object - Base Address** Format: SplitBaseAddress4KByteAligned Specifies the 4K-byte aligned memory base address for the write-only indirect data object pointed in the PAK_SLICE_STATE command for writing out the compressed bitstream. This field is only valid in AVC encoder mode.
23	31:0	**MFC Indirect PAK-BSE Object - Attributes** Format: MemoryAddressAttributes
24..25	63:0	**MFC Indirect PAK-BSE Object - Upper Bound** Format: SplitBaseAddress4KByteAligned This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the PAK_SLICE_STATE command for the per-slice output bitstream. Indirect data accessed at this address and beyond will be blocked by the hardware and ignored. Setting this field to 0 will cause this range check to be ignored If non-zero, this address must be greater than the MFC Indirect PAK-BSE Object Base Address state. This field is only valid in AVC encoder mode. **Programming Notes** For VP8 Encoder, this field should be programmed the same at both DW4, DW5 **MFX Indirect Bitstream Object - Access Upper Bound** as well as DW24, DW25.

MFX_JPEG_HUFF_TABLE_STATE

<table>
<tr><th colspan="3">MFX_JPEG_HUFF_TABLE_STATE</th></tr>
<tr><td colspan="3">Source: VideoCS
Length Bias: 2</td></tr>
<tr><td colspan="3">This Huffman table commands contains both DC and AC tables for either luma or chroma. Once a Huffman table has been defined for a particular destination, it replaces the previous tables stored in that destination and shall be used in the remaining Scans of the current image. A Huffman table will be sent to H/W only when it is loaded from bitstream.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline</b><table><tr><td>Default Value:</td><td>2h MFX_MULTI_DW</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><b>Media Command Opcode</b><table><tr><td>Default Value:</td><td>7h JPEG_COMMON</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>23:21</td><td><b>SubOpcode A</b><table><tr><td>Default Value:</td><td>0h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>20:16</td><td><b>SubOpcode B</b><table><tr><td>Default Value:</td><td>2h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b><table><tr><td>Default Value:</td><td>033Dh Excludes DWord (0,1)</td></tr><tr><td>Format:</td><td>=n</td></tr></table></td></tr>
<tr><td>1</td><td>31:1</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>HuffTableID (1-bit)</b>
Identifies the huffman table.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Y</td><td>Huffman table for Y</td></tr></table></td></tr>
</table>

MFX_JPEG_HUFF_TABLE_STATE		
2..4	95:0	**DC_BITS (12 8-bit array)** The number of DC Huffman codes of length i, where i is 1~12
5..7	95:0	**DC_HUFFVAL (12 8-bit array)** The value associated with each DC Huffman code of length i.
8..11	127:0	**AC_BITS (16 8-bit array)** the list of Li, number of Huffman codes of length i, where i is 1~16
12..51	1279:0	**AC_HUFFVAL (160 8-bit array)** the list of Vi,j, the value associated with each Huffman code of length i
52	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**AC_HUFFVAL(2-8 bit array)** In AC table, BITS can have up to 16-bit codeword. Li can be 0 ~ 162. HUFFVAL will have a list of likely random distributed values

MFX_JPEG_PIC_STATE

<table>
<tr><th colspan="3">MFX_JPEG_PIC_STATE</th></tr>
<tr><td colspan="3">Source: VideoCS
Length Bias: 2</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="7">0</td><td>31:29</td><td><b>Command Type</b>
<table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>28:27</td><td><b>Pipeline</b>
<table><tr><td>Default Value:</td><td>2h MFX_MULTI_DW</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>26:24</td><td><b>Media Command Opcode</b>
<table><tr><td>Default Value:</td><td>7h JPEG</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>23:21</td><td><b>SubOpcode A</b>
<table><tr><td>Default Value:</td><td>0h Common</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>20:16</td><td><b>SubOpcode B</b>
<table><tr><td>Default Value:</td><td>0h MEDIA_</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>15:12</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>11:0</td><td><b>DWord Length</b>
<table><tr><td>Format:</td><td>=n</td></tr></table>
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0001h</td><td><b>[Default]</b></td><td>Excludes DWord (0,1)</td></tr></table></td></tr>
<tr><td rowspan="2">1</td><td>31</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>30:26</td><td><b>Pixels In Horizontal Last MCU</b>
<table><tr><td>Exists If:</td><td>//Encoder Only</td></tr></table>
The number of pixels in the last MCU in a row MCUs. This information is used for completion of partial MCU.</td></tr>
</table>

<table>
<tr><th colspan="3">MFX_JPEG_PIC_STATE</th></tr>
<tr><td></td><td>25:21</td><td><b>Pixels In Vertical Last MCU</b>
<table><tr><td>Exists If:</td><td>//Encoder Only</td></tr></table>
The number of pixels in the last MCU in a column MCUs. This information is used for completion of partial MCU.</td></tr>
<tr><td></td><td>20</td><td><b>Vertical Up-Sampling Enable</b>
<table><tr><td>Exists If:</td><td>//Decoder Only</td></tr></table>
Only applied to chroma blocks. This flag is used for 2:1 vertical up-sampling for chroma 420 and outputting chroma422 YUY2 or UYVY format. To enable this flag, the input should be interleaved Scan, <b>InputFormatYUV</b> should be set to YUV420, and <b>OutputFormatYUV</b> should be set to YUY2 or UYVY.
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td></td><td>no up-sampling</td></tr>
<tr><td>1b</td><td></td><td>2:1 vertical up-sampling</td></tr></table></td></tr>
<tr><td></td><td>19</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>18</td><td><b>Horizontal Down-Sampling Enable</b>
<table><tr><td>Exists If:</td><td>//Decoder Only</td></tr></table>
Only applied to chroma blocks. This flag is used for 2:1 horizontal down-sampling for chroma 422 and outputting chroma420 NV21 format. To enable this flag, the input should be interleaved Scan, <b>InputFormatYUV</b> should be set to YUV422V_2Y or YUV422V_4Y, and <b>OutputFormatYUV</b> should be set to NV12.
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td></td><td>no down-sampling</td></tr>
<tr><td>1b</td><td></td><td>2:1 horizontal down-sampling</td></tr></table></td></tr>
<tr><td></td><td>17</td><td><b>Vertical Down-Sampling Enable</b>
<table><tr><td>Exists If:</td><td>//Decoder Only</td></tr></table>
Only applied to chroma blocks. This flag is used for 2:1 vertical down-sampling for chroma 422 and outputting chroma420 NV21 format. To enable this flag, the input should be interleaved Scan, <b>InputFormatYUV</b> should be set to YUV422H_2Y or YUV422H_4Y, and <b>OutputFormatYUV</b> should be set to NV12.
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td></td><td>no down-sampling</td></tr>
<tr><td>1b</td><td></td><td>2:1 vertical down-sampling</td></tr></table></td></tr>
<tr><td></td><td>16</td><td><b>Average Down Sampling</b>
<table><tr><td>Exists If:</td><td>//Decoder Only</td></tr></table>
This flag is used to select a down-sampling method when <b>VertDownSamplingEnb</b> or <b>HoriDownSamplingEnb</b> is set to 1.
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td></td><td>Drop every other line (or column) pixels</td></tr></table></td></tr>
</table>

MFX_JPEG_PIC_STATE

1b		Average neighboring two pixels

15:12 **Reserved**

Access:	RO
Format:	MBZ

11:8 **Input Surface Format YUV**

Exists If:	//Encoder Only

This field specifies the surface format to read a YUV image data

Value	Name	Description
0000b		Reserved
0001b	NV12	NV12 for chroma 4:2:0
0010b	UYVY	UYVY for chroma 4:2:2
0011b	YUY2	YUY2 for chroma 4:2:2
0100b	Y8	Y8 for chroma400 Y-only image
0101b	RGB	RGB or YUV for chroma 4:4:4

Programming Notes
This field should be set accordingly for **SurfaceFormat** in MFX_SURFACE_STATE command.
R8G8B8A8_UNORM in this field is used for encoding RGB and YUV chroma 4:4:4. For RGB input, any order of image components R, G, B (e.g., RGB, GBR, BGR, YUV) will be acceptable as far as the order of Quantization tables and Huffman tables match the order of image components.

11:8 **Output Format YUV**

Exists If:	//Decoder Only

This field specifies the surface format to write the decoded JPEG image. Note that any non-interleaved JPEG input should be set to "0000". For the interleaved input Scan data, it can be set either "0000" or the corresponding format.

Value	Name	Description
0000b		3 separate plane for Y, U, and V respectively
0001b		NV12 for chroma 4:2:0
0010b		UYVY for chroma 4:2:2
0011b		YUY2 for chroma 4:2:2

Programming Notes
The **MFX_SURFACE_STATE** command should be set accordingly for each **OutputFormatYUV**. For NV12, **Surface Format** = 4 (PLANAR_420_8) For YUY2, **Surface Format** = 0 (YCRCB_NORMAL) For UYVY, **Surface Format** = 3 (YCRCB_SWAPY) NV12 (0001b) can be set only when Y, U, V are interleaved in a single Scan data with the following cases • **InputFormatYUV** is YUV420 and **VertDownSamplingEnb** is disabled

		- **InputFormatYUV**is YUV422H_2Y or YUV422H_4Y, and **VertDownSamplingEnb** is enabled UYVY (0010b) and YUY2 (0011b) can be set only when Y, U, V are interleaved in a single Scan data with the following cases - **InputFormatYUV** is YUV420 and **VertUpSamplingEnb** is enabled - **InputFormatYUV** is YUV422H_2Y or YUV422H_4Y and **VertUpSamplingEnb** is disabled
	7:6	**Reserved** Access: RO Format: MBZ
	5:4	**Rotation** Exists If: //Decoder Only
	3	**Reserved** Access: RO Format: MBZ
	2:0	**Input Format YUV** Exists If: //Decoder Only Format: U3

Rotation

Exists If:	//Decoder Only

Value	Name	Description
00b		no rotation
01b		rotate clockwise 90 degree
10b		rotate counter-clockwise 90 degree (same as rotating 270 degree clockwise)
11b		rotate 180 degree (NOT the same as flipped on the x-axis)

Programming Notes
Rotation can be set to 01b, 10b, or 11b when OutputFormatYUV is set to 0000b. For other OutputFormatYUV, Rotation is not allowed.

Input Format YUV

Value	Name	Description
0	**[Default]**	YUV400 (grayscale image)
1		YUV420
2		YUV422H_2Y (Horizontally chroma 2:1 subsampled) - horizontal 2 Y-block, 1U and 1V
3		YUV444
4		YUV411

MFX_JPEG_PIC_STATE

5		YUV422V_2Y (Vertically chroma 2:1 subsampled) - vertical 2 Y-blocks, 1U and 1V
6		YUV422H_4Y - 2x2 Y-blocks, vertical 2U and 2V
7		YUV422V_4Y - 2x2 Y-blocks, horizontal 2U and 2V

2:0 **Output MCU Structure**

Exists If:	//Encoder Only

Output MCU Structure(**OutputMcuStructure**) should be set accordingly for each Input Surface Format YUV(**InputSurfaceFormatYUV**):

- If **InputSurfaceFormatYUV** is set to NV12, **OutputMCUStructure** is set to YUV420.
- If **InputSurfaceFormatYUV** is set to UYVY or YUY2, **OutputMCUStructure** is set to YUV422H_2Y.
- If **InputSurfaceFormatYUV** is set to Y8, **OutputMCuStructure** is set to YUV400.
- If **InputSurfaceFormatYUV** is set to RGB (or GBR, BGR, YUV), **OutputMCuStructure** is set to RGB.
- If **InputSurfaceFormatYUV** is set to RGB, the order of encoded blocks in MCU will be same as the order of input image components. If the order of input image components is RGB (or GBR, BGR, YUV), then the order of blocks will be RGB (or GBR, BGR, YUV respectively).

Value	Name	Description
0	YUV400	Grayscale Image
1	YUV420	Both horizontally and vertically chroma 2:1 subsampled
2	YUV422H_2Y	Horizontally chroma 2:1 subsampled - horizontal 2 Y-blocks, 1 U and 1 V block
3	RGB	RGB or YUV444: No subsample
4		
5		
6		
7		

2 31:30 **Reserved**

Access:	RO
Format:	MBZ

29 **Output Pixel Normalize**

Exists If:	//Decoder Only

JPEG decoded output pixels for Y and U/V in order to adjust display YUV range.

Value	Name	Description	Exists If
0		No Normalization	
1		Normalize output pixels from [0,255] to [16,235]	//Y
1		Normalize output pixels from [0,255] to [16,239]	//U/V

<table>
<tr><th colspan="3">MFX_JPEG_PIC_STATE</th></tr>
<tr><td></td><td>28:16</td><td><b>Frame Height In Blocks Minus 1</b>
<table>
<tr><td>Exists If:</td><td>//Decoder Only</td></tr>
<tr><td>Format:</td><td>U13-1</td></tr>
</table>
(The number of blocks in height) - 1.This value is calculated using the number of lines Y and vertical sampling factor of the first component V_1 in Frame header. See the note following this table. For interleaved components, (((Y + (V_1*8 -1)) / (V_1*8)) * V_1) - 1, where "/" is integer division. For non-interleaved components, ((Y + 7) / 8) - 1.</td></tr>
<tr><td></td><td>28:16</td><td><b>Frame Height In Blks Minus 1</b>
<table>
<tr><td>Exists If:</td><td>//Encoder Only</td></tr>
<tr><td>Format:</td><td>U13-1</td></tr>
</table>
<table>
<tr><td>(The number of blocks in height) - 1. This value is calculated using the number of lines Y and vertical sampling factor of the first component V1 in Frame header. See the note following this table.</td></tr>
<tr><td>For interleaved components: (((Y + (V1*8 -1)) / (V1*8)) * V1) - 1For non-interleaved components: ((Y + 7) / 8) - 1</td></tr>
</table></td></tr>
<tr><td></td><td>15:13</td><td><b>RoundingQuant</b>
<table>
<tr><td>Exists If:</td><td>//Encoder Only</td></tr>
</table>
Rounding value applied to quantization output
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>000b</td><td><b>[Default]</b></td><td>1/2</td></tr>
<tr><td>001b</td><td></td><td>(1/2 - 1/128)</td></tr>
<tr><td>010b</td><td></td><td>(1/2 + 1/128)</td></tr>
<tr><td>011b</td><td></td><td>(1/2 - 1/64)</td></tr>
<tr><td>100b</td><td></td><td>(1/2 + 1/64)</td></tr>
<tr><td>101b</td><td></td><td>(1/2 - 1/32)</td></tr>
<tr><td>110b</td><td></td><td>(1/2 - 1/16)</td></tr>
<tr><td>111b</td><td></td><td>(1/2 - 1/8)</td></tr>
</table></td></tr>
<tr><td></td><td>12:0</td><td><b>Frame Width In Blocks Minus 1</b>
<table>
<tr><td>Exists If:</td><td>//Decoder Only</td></tr>
<tr><td>Format:</td><td>U13-1</td></tr>
</table>
(The number of blocks in width) - 1. This value is calculated using the number of samples per line X and horizontal sampling factor of the first component H_1 in Frame header. See the note following this table. For interleaved components, (((X + (H_1 *8 -1)) / (H_1 *8)) * H_1) - 1.For non-interleaved components, ((X + 7) / 8) - 1.</td></tr>
<tr><td></td><td>12:0</td><td><b>Frame Width In Blks Minus 1</b>
<table>
<tr><td>Exists If:</td><td>//Encoder Only</td></tr>
<tr><td>Format:</td><td>U13-1</td></tr>
</table></td></tr>
</table>

MFX_JPEG_PIC_STATE		
		(The number of blocks in width) - 1. This value is calculated using the number of samples per line X and horizontal sampling factor of the first component H1 in Frame header. See the note following this table. For interleaved components: (((X + (H1 *8 -1)) / (H1 *8)) * H1) – 1 For non-interleaved components: ((X + 7) / 8) - 1

MFX_MPEG_TS_CONTROL command

MFX_MPEG_TS_CONTROL command		
Source:	VideoCS	
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h Command Type Format: Opcode
	28:27	**Pipeline** Default Value: 2h Format: Opcode
	26:24	**Opcode** Default Value: 0h Format: Opcode
	23:21	**SubOpA** Default Value: 2h Format: Opcode
	20:16	**SubOpB** Default Value: Bh Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** **Value** | **Name** | **Description** 3h | DWORD_COUNT_n **[Default]** | Total length - 2 (excludes DWord0 and DWord1)
1	31:30	**Reserved** Access: RO Format: MBZ
	29	**payload_unit_start_indicator control** **Programming Notes** This bit should be programmed zero always.
	28	**Additional Copy info flag in PES header**
	27	**DSM trick mode flag in PES header**
	26	**Original or flag in PES header**
	25	**Copy Right flag in PES header**

MFX_MPEG_TS_CONTROL command

	24	**Output TS packet grouping select** 0: Return all packets continuously 1: Return 7 packets in 2K aligned buffer (with the remaining bits between the end of the 7th packet and the end of the 2K buffer including the rest being undefined)
	23:20	**StreamID lower Nibble** Stream ID Lower Nibble. Stream ID for Video can be 0xE0 through 0xEF. This 4 bit field indicates the last 4 bits of Stream ID in Mpeg transport stream as indicated in the DCN diagram
	19:13	**Reserved** Access: RO Format: MBZ
	12:0	**Video PacketID Header Parameter** This field specifies the static program fields in MPEG header for each Video packet.
2	31:0	**PCR 90 KHz component least significant bits.**
3	31:23	**27MHz Counter** Full 8-bits of 27Mhz counter
	22:1	**Reserved** Access: RO Format: MBZ
	0	**90 KHz counter MSB** Upper bit (bit 33) of 90khz counter
4	31:0	**PTS Delta** PTS Delta to be applied to 90 KHz count of PCR to generate PTS. This is a Twos complement number and added to 90 KHz PCR counter to generate PTS.
5	31:28	**Continuity Counter** This field specifies the 4b continuity counter given in the MPEGTS packet header. This should be initialized with the value that was read from MMIO at the end of the previous frame. That value will be incremented by HW so there is no need to SW to increment it. For the first frame this should be set to 0.
	27:16	**Reserved** Access: RO Format: MBZ
	15:0	**MPEGTS Packet Count** This field is ignored by HW. Driver can copy MFX_PAK_MPEGTS_STATUS register from the previous frame to DW 5 of MPEG_TS_CONTROL_command using MI_STORE_REG_MEM

MFX_MPEG2_PIC_STATE

MFX_MPEG2_PIC_STATE		
Source:	VideoCS	
Length Bias:	2	

This must be the very first command to issue after the surface state, the pipe select and base address setting commands. For MPEG-2 the encoder is called per slice-group, however the picture state is called per picture. Notice that a slice-group is a group of consecutive slices that no non-trivial slice headers are inserted in between.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MPEG2_PIC_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 3h MPEG2_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 0h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 0h Excludes DWord (0,1)= 00Bh, used for normal decode and encode mode000h, a special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware. Format: =n
1	31:28	**f_code[1][1].** Used for backward motion vector prediction. See ISO/IEC 13818-2 7.6.3.1 for details
	27:24	**f_code[1][0].** Used for backward motion vector prediction. See ISO/IEC 13818-2 7.6.3.1 for details

<table>
<tr><th colspan="3">MFX_MPEG2_PIC_STATE</th></tr>
<tr><td></td><td>23:20</td><td><b>f_code[0][1]</b>
Used for forward motion vector prediction. See ISO/IEC 13818-2 7.6.3.1 for details</td></tr>
<tr><td></td><td>19:16</td><td><b>f_code[0][0]</b>
Used for forward motion vector prediction. See ISO/IEC 13818-2 7.6.3.1 for details</td></tr>
<tr><td></td><td>15:14</td><td><b>Intra DC Precision</b><table><tr><td>Format:</td><td>U2</td></tr></table>See ISO/IEC 13818-2 6.3.10 for details.</td></tr>
<tr><td></td><td>13:12</td><td><b>Picture Structure</b>
This field specifies whether the picture is encoded in the form of a frame picture or one field (top or bottom) picture. See ISO/IEC 13818-2 6.3.10 for details. Format = MPEG_PICTURE_STRUCTURE00 = Reserved01 = MPEG_TOP_FIELD10 = MPEG_BOTTOM_FIELD11 = MPEG_FRAME</td></tr>
<tr><td></td><td>11</td><td><b>TFF (Top Field First)</b>
When two fields are stored in a picture, this bit indicates if the top field is the first field. For a frame P picture, the value 1 indicates that the top field of the reconstructed frame is the first field output by the decoding process, the same as defined in ISO/IEC 13818-2 6.3.10. Particularly, it is used by the hardware to calculate derivative motion vectors from the dual-prime motion vectors. For a field P picture, hardware uses this bit together with the Picture Structure to determine if the current picture is the Second Field. In this case, the definition of this bit differs from ISO/IEC 13818-2 6.3.10 - software must derive the value for this bit according to the following relation: Picture Structure = top field Picture Structure = bottom field Second Field = 0TFF = 1TFF = 0Second Field = 1TFF = 0TFF = 1</td></tr>
<tr><td></td><td>10</td><td><b>Frame Prediction Frame DCT</b>
This field provides constraints on the DCT type and prediction type. It affects the syntax of the bitstream.</td></tr>
<tr><td></td><td>9</td><td><b>Concealment Motion Vector Flag</b>
This field indicates if the concealment motion vectors are coded in intra macroblocks. It affects the syntax of the bitstream.</td></tr>
<tr><td></td><td>8</td><td><b>Quantizer Scale Type</b><table><tr><td>Format:</td><td>Boolean</td></tr></table>This field specifies the quantizer scaling type.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td></td><td>MPEG_QSCALE_LINEAR</td></tr><tr><td>1h</td><td></td><td>D MPEG_QSCALE_NONLINEAR esc</td></tr></table></td></tr>
<tr><td></td><td>7</td><td><b>Intra VLC Format</b>
This field is used by VLD</td></tr>
<tr><td></td><td>6</td><td><b>Scan Order</b><table><tr><td>Format:</td><td>Boolean</td></tr></table>This field specifies the Inverse Scan method for the DCT-domain coefficients in the blocks of the current picture.</td></tr>
</table>

<table>
<tr><th colspan="3">MFX_MPEG2_PIC_STATE</th></tr>
<tr><td></td><td></td><td><table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td></td><td>MPEG_ZIGZAG_SCAN</td></tr>
<tr><td>1h</td><td></td><td>MPEG_ALTERNATE_VERTICAL_SCAN</td></tr>
</table></td></tr>
<tr><td></td><td>5:0</td><td><b>Reserved</b><table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>2</td><td>31</td><td><b>I Slice Concealment Mode</b><table>
<tr><td>Exists If:</td><td>//Decoder</td></tr>
</table>This field controls how MPEG decoder handles MB concealment in I Slice<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Intra Concealment</td><td>Using Coefficient values to handle MB concealment</td></tr>
<tr><td>1h</td><td>Inter Concealment</td><td>Using Motion Vectors to handle MB concealment</td></tr>
</table><table>
<tr><th>Programming Notes</th></tr>
<tr><td>If this field is set to "1", driver must provide a valid forward reference picture (both top and bottom Field must be valid)</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b><table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>29:28</td><td><b>P/B Slice Concealment Mode</b><table>
<tr><td>Exists If:</td><td>//Decoder</td></tr>
</table>This field controls how MPEG decoder handles MB concealment in P/B Slice.<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>00b</td><td>INTER</td><td>If left MB is NOT Intra MB type (including skip MB), use left MB inter prediction mode [frame/field or forward/backward/bi] and MV final values as concealment. Otherwise (left MB is Intra MB), use forward reference (same polarity for field pic) with MV final values set to 0.</td></tr>
<tr><td>01b</td><td>LEFT</td><td>If left MB is NOT Intra MB type (including skip MB), use left MB inter prediction mode [frame/field or forward/backward/bi] and MV final values as concealment. Otherwise (left MB is Intra MB), use left MB dct_dc_pred[cc] values for concealment (Macroblock is concealed as INTRA MB and dct_dc_pred[cc] are DC predictor for Luma, Cr, Cb data)</td></tr>
<tr><td>10b</td><td>ZERO</td><td>Always use forward reference (same polarity for field pic) with MV final values set to 0 (Macroblock is concealed as INTER coded)</td></tr>
<tr><td>11b</td><td>INTRA</td><td>Use left MB dct_dc_pred[cc] values for concealment (Macroblock is concealed as INTRA MB and dct_dc_pred[cc] are DC predictor for Luma, Cr, Cb data</td></tr>
</table></td></tr>
</table>

MFX_MPEG2_PIC_STATE

27

Reserved

Access:	RO
Format:	MBZ

26:25

P/B Slice Predicted BiDir Motion Type Override - Bi-direction MV Type Override

Exists If:	//Decoder

This field is only applicable if the Concealment Motion Type is predicted to be Bi-directional. (It is only possible if "P/B Slice Concealment Mode" is set to "00" or "01" and left MB is a bi-directional MB).

Value	Name	Description
0h	BID	Keep Bi-direction Prediction
1h	RESERVED	
2h	FWD	Only use Forward Prediction (Backward MV is forced to invalid
3h	BWD	Only use Backward Prediction (Forward MV is forced to invalid)

24

P/B Slice Predicted Motion Vector Override Final MV value Override

Exists If:	//Decoder

This field is only applicable if the Concealment Motion Vectors are non-zero. It is only possible if "P/B Slice Concealment Mode" is set to "00" or "01" and left MB has non-zero motion vectors).

Value	Name	Description
0h	Predicted	Motion Vectors use predicted values
1h	ZERO	Motion Vectors force to 0

23:15

Reserved

Access:	RO
Format:	MBZ

14

LoadSlicePointerFlag - LoadBitStreamPointerPerSlice

Exists If:	//Encoder

To support multiple slice picture and additional header/data insertion before and after an encoded slice. When this field is set to 0, bitstream pointer is only loaded once for the first slice of a frame. For subsequent slices in the frame, bitstream data are stitched together to form a single output data stream. When this field is set to 1, bitstream pointer is loaded for each slice of a frame. Basically, bitstream data for different slices of a frame will be written to different memory locations.

Value	Name	Description
0h		Load BitStream Pointer only once for the first slice of a frame
1h		Load/reload BitStream Pointer only once for each slice, reload the start location of the bitstream buffer from the Indirect PAK-BSE Object Data Start Address field

<table>
<tr><th colspan="3">MFX_MPEG2_PIC_STATE</th></tr>
<tr><td></td><td>13:11</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>10:9</td><td><b>Picture Coding Type</b>
<table>
<tr><td>Format:</td><td>U2</td></tr>
</table>
This field identifies whether the picture is an intra-coded picture (I), predictive-coded picture (P) or bi-directionally predictive-coded picture (B). See ISO/IEC 13818-2 6.3.9 for details.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>Reserved</td></tr>
<tr><td>01b</td><td>MPEG_I_PICTURE</td></tr>
<tr><td>10b</td><td>10 = MPEG_P_PICTURE</td></tr>
<tr><td>11b</td><td>MPEG_B_PICTURE</td></tr>
</table></td></tr>
<tr><td></td><td>8:2</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>1:0</td><td><b>MismatchControlDisabled</b>
These 2 bits flag disables mismatch control of the inverse transformation for some specific cases during reference reconstruction. To disable MPEG2 IDCT fixed point arithmetic correction.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>00b</td><td></td><td>Mismatch control applies to all MBs</td></tr>
<tr><td>01b</td><td></td><td>Disable mismatch control to all intra MBs whose all AC-coefficients are zero.</td></tr>
<tr><td>10b</td><td></td><td>Disable mismatch control to all MBs whose all AC-coefficients are zero.</td></tr>
<tr><td>11b</td><td></td><td>Disable mismatch control to all MBs.</td></tr>
</table></td></tr>
<tr><td>3</td><td>31</td><td><b>Slice Concealment Disable Bit</b>
<table>
<tr><td>Exists If:</td><td>//Decode</td></tr>
</table>
If VINunit detects the next slice starting position is either out-of-bound or smaller than or equal to the current slice starting position, VIN will set the current slice to be 1 MB and force VMDunit to do slice concealment on the next slice. This bit will disable this feature and the MB data from the next slice will be decoded from bitstream.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Enable <b>[Default]</b></td><td>VIN will force next slice to be concealment if detects slice boundary error</td></tr>
<tr><td>1h</td><td>Disable</td><td>VIN will not force next slice to be in concealment</td></tr>
</table></td></tr>
</table>

MFX_MPEG2_PIC_STATE

		Programming Notes Driver has an option to detect the scenario given in description (above) and remove the second (out-of-order) slice. In this case, hardware will decode the first slice in completion and do concealment till the third slice. It should yield a picture with better quality this way.
	30:29	**Reserved** Access: RO Format: MBZ
	28:24	**Reserved**
	23:16	**FrameHeightInMBsMinus1[7:0] (Picture Height in Macroblocks)** Format: U8
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**FrameWidthInMBsMinus1[7:0] (Picture Width in Macroblocks)** Format: U8
4	31:16	**MinFrameWSize** Format: U16 Minimum Frame Size [15:0] (16-bit) (Encoder Only) Minimum Frame Size is specified to compensate for intel Rate ControlCurrently zero fill (no need to perform emulation byte insertion) is done only to the end of the CABAC_ZERO_WORD insertion (if any) at the last slice of a picture. Intel encoder parameter. The caller should always make sure that the value, represented by Minimum Frame Size, is always less than maximum frame size FrameBitRateMax (DWORD 10 bits 29:16). This field is reserved in Decode mode. **Programming Notes** Programmable range is 0..(2^16-1).
	15	**Reserved** Access: RO Format: MBZ
	14:12	**RoundInterAC,** rounding precision for non-Intra AC000: +1/16001: +2/16010: +3/16011: +4/16100: +5/16101: +6/16110: +7/16111: +8/16
	11	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">MFX_MPEG2_PIC_STATE</th></tr>
<tr><td></td><td>10:8</td><td><b>RoundIntraAC</b><table><tr><td>Format:</td><td>U3</td></tr></table>rounding precision for Intra AC000: +1/16001: +2/16010: +3/16011: +4/16100: +5/16101: +6/16110: +7/16111: +8/16</td></tr>
<tr><td></td><td>7</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>6:4</td><td><b>RoundInterDC</b>
rounding Precision for non-Intra-DC000: +1/16001: +2/16010: +3/16011: +4/16100: +5/16101: +6/16110: +7/16111: +8/16</td></tr>
<tr><td></td><td>3</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>2:1</td><td><b>RoundIntraDC</b>
rounding Precision for Intra-DC00: +1/801: +2/810: +3/811: +4/8</td></tr>
<tr><td></td><td>0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>5</td><td>31:17</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>16</td><td><b>FrameSizeControlMask</b>
Frame size conformance mask This field is used when MacroblockStatEnable is set to 1.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td></td><td>Do not change Slice Quantization Parameter values in MFC_MPEG2_SLICEGROUP_STATE with suggested slice QP value for frame level Rate control</td></tr><tr><td>1h</td><td></td><td>Replace Slice Quantization Parameter values in MFC_MPEG2_SLICEGROUP_STATE with suggested slice QP value for frame level Rate control values in MFC_IMAGE_STATUS control register.</td></tr></table></td></tr>
<tr><td></td><td>15:13</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>12</td><td><b>InterMBForceCBPZeroControlMask</b><table><tr><td>Format:</td><td>U1</td></tr></table>Inter MB Force CBP ZERO mask.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td></td><td>No effect</td></tr></table></td></tr>
</table>

MFX_MPEG2_PIC_STATE

1h		Zero out all A/C coefficients for the inter MB violating Inter Conformance

11:10 **MinFrameWSizeUnits**

This field is the Minimum Frame Size Units

Value	Name	Description
00b	compatibility mode	Minimum Frame Size is in old mode (words, 2bytes)
01b	16 byte	Minimum Frame Size is in 16bytes
10b	4Kb	Minimum Frame Size is in 4Kbytes
11b	16Kb	Minimum Frame Size is in 16Kbytes

9 **MBRateControlMask**

MB Rate Control conformance mask This field is ignored when MacroblockStatEnable is disabled or MB level Rate control flag for the current MB is disable in Macroblock Status Buffer.

Value	Name	Description
0h		Do not change QP values of inter macroblock with suggested QP values in Macroblock Status Buffer
1h		Apply RC QP delta for all macroblock

8:4 **Reserved**

Access:	RO
Format:	MBZ

3 **FrameBitRateMinReportMask**

This is a mask bit controlling if the condition of frame level bit count is less than FrameBitRateMin.

Value	Name	Description
0h	Disable	Do not update bit0 of MFC_IMAGE_STATUS control register.
1h	Enable	set bit0 and bit 1of MFC_IMAGE_STATUS control register if the total frame level bit counter is less than or equal to Frame Bit rate Minimum limit.

2 **FrameBitRateMaxReportMask**

This is a mask bit controlling if the condition of frame level bit count exceeds FrameBitRateMax.

Value	Name	Description
0h	Disable	Do not update bit0 of MFC_IMAGE_STATUS control register.
1h	Enable	set bit0 and bit 1 of MFC_IMAGE_STATUS control register if the total frame level bit counter is greater than or equal to Frame Bit rate Maximum limit.

1 **InterMBMaxSizeReportMask**

This is a mask bit controlling if the condition of any inter MB in the frame exceeds InterMBMaxSize.

<table>
<tr><th colspan="3">MFX_MPEG2_PIC_STATE</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td></td><td>Do not update bit0 of MFC_IMAGE_STATUS control register.</td></tr><tr><td>1h</td><td></td><td>set bit0 of MFC_IMAGE_STATUS control register if the total bit counter for the current MB is greater than the Inter MB Conformance Max size limit.</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>IntraMBMaxSizeReportMask</b>
This is a mask bit controlling if the condition of any intra MB in the frame exceeds IntraMBMaxSize.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td></td><td>Do not update bit0 of MFC_IMAGE_STATUS control register.</td></tr><tr><td>1h</td><td></td><td>set bit0 of MFC_IMAGE_STATUS control register if the total bit counter for the current MB is greater than the Intra MB Conformance Max size limit.</td></tr></table></td></tr>
<tr><td>6
[ExistsIf]Encode Only</td><td>31:28</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>27:16</td><td><b>InterMBMaxSize</b><table><tr><td>Default Value:</td><td>FFFh</td></tr></table>This field, Inter MB Conformance Max size limit, indicates the allowed max bit count size for Inter MB</td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>IntraMBMaxSize</b><table><tr><td>Default Value:</td><td>FFFh</td></tr></table>This field, Intra MB Conformance Max size limit,indicates the allowed max bit count size for Intra MB</td></tr>
<tr><td>7</td><td>31:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>8
[ExistsIf]Encode Only</td><td>31:24</td><td><b>SliceDeltaQPMax[3]</b><table><tr><td>Format:</td><td>S7</td></tr></table><table><tr><td>This field is the Slice level delta QP for total bit-count above FrameBitRateMax - first 1/8 region This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame exceeds FrameBitRateMax but is within 1/8of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of(FrameBitRateMax, (FrameBitRateMax+FrameBitRateMaxDelta»3).</td></tr><tr><td>Range: [-30,30]</td></tr></table></td></tr>
</table>

MFX_MPEG2_PIC_STATE

<table>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>0h</td><td>Disable</td></tr><tr><td>1h</td><td>Enable</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>SliceDeltaQPMax[2]</b><table><tr><td>Format:</td><td>S7</td></tr></table><table><tr><td>Range: [-30,30]</td></tr><tr><td>This field is the Slice level delta QP forbit-count above FrameBitRateMax - above 1/8 and below 1/ 4 This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between 1/8 and of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of((FrameBitRateMax+ FrameBitRateMaxDelta»3), (FrameBitRateMax+FrameBitRateMaxDelta»2).</td></tr></table></td></tr>
<tr><td></td><td>15:8</td><td><b>SliceDeltaQPMax[1]</b><table><tr><td>Format:</td><td>S7</td></tr></table><table><tr><td>Range: [-30,30]</td></tr><tr><td>This field is the Slice level delta QP forbit-count above FrameBitRateMax - above1/ 4 and below 1/2 This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between and of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of ((FrameBitRateMax+FrameBitRateMaxDelta»2), (FrameBitRateMax+FrameBitRateMaxDelta»1).</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>SliceDeltaQPMax[0]</b><table><tr><td>Format:</td><td>S7</td></tr></table><table><tr><td>Range: [-30,30]</td></tr><tr><td>This field is the Slice level delta QP forbit-count above FrameBitRateMax - above 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bitcount for the entire frame is above FrameBitRateMax by more than half the distance of FrameBitRateMaxDelta , i.e., in the range of ((FrameBitRateMax+FrameBitRateMaxDelta»1), infinite).</td></tr></table></td></tr>
<tr><td>9
[ExistsIf]Encode Only</td><td>31:24</td><td><b>SliceDeltaQPMin[3]</b><table><tr><td>Format:</td><td>S7</td></tr></table><table><tr><td>Range: [-30,30]</td></tr><tr><td>This field is the Slice level delta QP for total bit-count below FrameBitRateMin - first 1/8 region This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is less than FrameBitRateMin and greater than or equal to 1/8 the distance of FrameBitRateMinDelta from FrameBitRateMin,i.e., in the range of [(FrameBitRateMin-</td></tr></table></td></tr>
</table>

MFX_MPEG2_PIC_STATE

<table>
<tr><td></td><td></td><td>FrameBitRateMinDelta»3), FrameBitRateMin).</td></tr>
<tr><td></td><td>23:16</td><td><b>SliceDeltaQPMin[2]</b><table><tr><td>Format:</td><td>S7</td></tr></table><table><tr><td>Range: [-30,30]</td></tr><tr><td>This field is the Slice level delta QP forbit-count below FrameBitRateMin - below 1/ 8 and above 1/ 4This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between one-eighth and quarter the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of[(FrameBitRateMin- FrameBitRateMinDelta»2), (FrameBitRateMin-FrameBitRateMinDelta»3)).</td></tr></table></td></tr>
<tr><td></td><td>15:8</td><td><b>SliceDeltaQPMin[1]</b><table><tr><td>Format:</td><td>S7</td></tr></table><table><tr><td>Range: [-30,30]</td></tr><tr><td>This field is the Slice level delta QP forbit-count below FrameBitRateMin- below 1/4 and above 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between quarter and half the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of[(FrameBitRateMin- FrameBitRateMinDelta»1), (FrameBitRateMin-FrameBitRateMinDelta»2)).</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>SliceDeltaQPMin[0]</b><table><tr><td>Format:</td><td>S7</td></tr></table><table><tr><td>Range: [-30,30]</td></tr><tr><td>This field is the Slice Level Delta QP forbit-count below FrameBitRateMin - below 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bitcount for the entire frame is below FrameBitRateMin by more than half the distance of FrameBitRateMinDelta , i.e., in the range of [0, (FrameBitRateMin-FrameBitRateMinDelta»1).</td></tr></table></td></tr>
<tr><td>10
[ExistsIf]Encode Only</td><td>31</td><td><b>FrameBitrateMaxUnit</b>
This field is the Frame Bitrate Maximum Limit Units.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Byte</td><td>FrameBitRateMax is in units of 32 Bytes when FrameBitrateMaxUnitMode is 1 and in units of 128 Bytes if FrameBitrateMaxUnitMode is 0</td></tr><tr><td>1h</td><td>Kilobyte</td><td>FrameBitRateMax is in units of 4KBytes Bytes when FrameBitrateMaxUnitMode is 1 and in units of 16KBytes if FrameBitrateMaxUnitMode is 0</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>FrameBitrateMaxUnitMode</b>
BitFiel This field is the Frame Bitrate Maximum Limit Units.dDesc</td></tr>
</table>

MFX_MPEG2_PIC_STATE

Value	Name	Description
0h	Compatibility mode	FrameBitRateMaxUnit is in old mode (128b/16Kb)
1h	New mode	FrameBitRateMaxUnit is in new mode (32byte/4Kb)

29:16 **FrameBitRateMax**

This field is the Frame Bitrate Maximum Limit. This field along with FrameBitrateMaxUnit determines maximum allowed bits in a frame before multi-pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count exceeds this value. When FrameBitrateMaxUnitMode is 0(compatibility mode) bits 16:27 should be used, bits 28 and 29 should be 0.

Value	Name	Description
[0-16383]		WhenFrameBitrateMaxUnit =0, this field is measured in unit of 32 bytes. The frame rate in bytes is range from 0-512KBytes, hence this field is programmed from 0 to 16,384 (14-bits) unit.
[0-16383]		WhenFrameBitrateMaxUnit =1, this field is measured in unit of 4K bytes (1K=1024). The frame rate in bytes is range from 0-64MBytes, hence the .this field is programmed from 0 to 16,384 (14-bits) unit.

15 **FrameBitrateMinUnit**

This field is the Frame Bitrate Minimum Limit Units.

Value	Name	Description
0h	Byte	FrameBitRateMax is in units of 32 Bytes when FrameBitrateMinUnitMode is 1 and in units of 128 Bytes if FrameBitrateMinUnitMode is 0
1h	KiloByte	FrameBitRateMax is in units of 4KBytes Bytes when FrameBitrateMaxUnitMode is 1 and in units of 16KBytes if FrameBitrateMaxUnitMode is 0

14 **FrameBitrateMinUnitMode**

This field is the Frame Bitrate Minimum Limit Units. ValueNameDescriptionProject

Value	Name	Description
0h	compatibility mode	FrameBitRateMaxUnit is in old mode (128b/16Kb)
1h	New Mode	FrameBitRateMaxUnit is in new mode (32byte/4Kb)

13:0 **FrameBitRateMin**

This field is the Frame Bitrate Minimum Limit. This field along with FrameBitrateMinUnit determines minimum allowed bits in a frame before multi-pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count happen to be below this value.

It takes on a value in the range of [0-16383].

When FrameBitrateMinUnitMode is 0 (compatibility mode) bits 0:11 should be used, bits 12 and 13 should be 0.

When FrameBitrateMinUnit=0, this field is measured in unit of 32 bytes. The frame rate in bytes is range from 0-512KBytes, hence the .this field is programmed from 0 to 16,384 (14-bits) unit.

When FrameBitrateMinUnit =1, this field is measured in unit of 4K bytes (1K=1024).The

<table>
<tr><th colspan="3">MFX_MPEG2_PIC_STATE</th></tr>
<tr><td></td><td></td><td>frame rate in bytes is range from 0-64MBytes, hence the .this field is programmed from 0 to 16,384 (14-bits) unit.</td></tr>
<tr><td rowspan="4">11
[ExistsIf]Encode Only</td><td>31</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>30:16</td><td><b>FrameBitRateMaxDelta</b>
<table><tr><td>Default Value:</td><td>0h</td></tr><tr><td>Access:</td><td>None</td></tr><tr><td>Format:</td><td>U15</td></tr></table>
<table><tr><td>This field is used to select the slice delta QP when FrameBitRateMax Is exceeded. It shares the same FrameBitrateMaxUnit. When FrameBitrateMaxUnitMode is 0(compatibility mode), only bits 16:27 should be used, bits 28, 29 and 30 should be 0.</td></tr>
<tr><td>This field is used to select the slice delta QP when FrameBitRateMax Is exceeded. It shares the same FrameBitrateMaxUnit. When FrameBitrateMaxUnitMode is 0(compatibility mode) bits 16:27 should be used, bits 28, 29 and 30 should be 0.
Range : [0-32767]
When FrameBitrateMaxUnit =0, this field is measured in unit of 32 bytes. The frame rate in bytes is range from 0-1024KBytes, hence the .this field is programmed from 0 to 32,767 (15-bits) unit.
When FrameBitrateMaxUnit =1, this field is measured in unit of 4K bytes (1K=1024).The frame rate in bytes is range from 0-128MBytes, hence the .this field is programmed from 0 to 32,767 (14-bits) unit.</td></tr></table></td></tr>
<tr><td>15</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>14:0</td><td><b>FrameBitRateMinDelta</b>
This field is used to select the slice delta QP when FrameBitRateMin Is exceeded. It shares the same FrameBitrateMinUnit. When FrameBitrateMinUnitMode is 0(compatibility mode) bits 0:11 should be used, bits12, 13 and 14 should be 0.Note: HW requires the following condition FrameBitRateMinDelta <= 2*FrameBitRateMinMust be true, otherwise it may cause unpredicted behavior.
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>[0-32767]</td><td></td><td>When FrameBitrateMaxUnit =0, this field is measured in unit of 32 bytes. The frame rate in bytes is range from 0-1024KBytes, hence .this field is programmed from 0 to 32,767 (15-bits) unit.</td></tr>
<tr><td>[0-32767]</td><td></td><td>When FrameBitrateMaxUnit =1, this field is measured in unit of 4K bytes (1K=1024).The frame rate in bytes is range from 0-128MBytes, hence the .this field is programmed from 0 to 32,767 (14-bits) unit.</td></tr></table></td></tr>
</table>

MFX_MPEG2_PIC_STATE		
12	31:0	**Reserved** Access: RO Format: MBZ

MFX_PAK_INSERT_OBJECT

MFX_PAK_INSERT_OBJECT

Source:	VideoCS
Length Bias:	2

The MFX_PAK_INSERT_OBJECT command is the first primitive command for the AVC, MPEG2, JPEG, and VP8 Encoding Pipeline.

This command is issued to setup the control and parameters of inserting a chunk of compressed/encoded bits into the current bitstream output buffer starting at the specified bit location to perform the actual insertion by transferring the command inline data to the output buffer max, 32 bits at a time.
It is a variable length command as the data to be inserted are presented as inline data of this command. It is a multiple of 32-bit (1 DW), as the data bus to the bitstream buffer is 32-bit wide.
Multiple insertion commands can be issued back-to-back in a series. It is host software's responsibility to make sure their corresponding data will properly stitch together to form a valid H.264 bitstream.
Internally, MFX hardware will keep track of the very last two bytes' (the very last byte can be a partial byte) values of the previous insertion. It is required that the next Insertion Object Command or the next PAK Object Command to perform the start code emulation sequence check and prevention 0x03 byte insertion with this end condition of the previous insertion.
Hardware will keep track of an output bitstream buffer current byte position and the associated next bit insertion position index. Data to be inserted can be a valid H.264 NAL units or a partial NAL unit. Certain NAL unit has a minimum byte size requirement. As such the hardware will optionally (enabled by STATE Command) determines the number of CABAC_ZERO_WORD to be inserted to the end of the current NAL, based on the minimum byte size of a NAL and the actual bin count of the encoded Slice. Since prior to the CABAC_ZERO_WORD insertion, the RBSP or EBSP is already byte-aligned, so each CABAC_ZERO_WORD insertion is actually a 3-byte sequence 0x00 00 03. The inline data may have already been processed for start code emulation byte insertion, except the possibility of the last 2 bytes plus the very last partial byte (if any). Hence, when hardware performing the concatenation of multiple consecutive insertion commands, or concatenation of an insertion command and a PAK object command, it must check and perform the necessary start code emulation byte insert at the junction. The inline data is required to be byte aligned on the left (first transmitted bit order) and may or may not be byte aligned on the right (last transmitted bits).

The command will specify the bit offset of the last valid DW. Each insertion state command defines a chunk of bits (compressed data) to be inserted at a specific location of the output compressed bitstream in the output buffer. Depend on CABAC or CAVLC encoding mode (from Slice State), PAK Object Command is always ended in byte aligned output bitstream except for CABAC header insertion which is bit aligned. In the aligned cases, PAK will perform 0 filling in CAVLC mode, and 1 filling in CABAC mode.
Insertion data can include: any encoded syntax elements bit data before the encoded Slice Data (PAK Object Command) of the current SliceSPS NALPPS NALSEI NAL Other Non-Slice NALLeading_Zero_8_bits (as many bytes as there is)Start Code Prefix NAL Header Byte Slice Header Any encoded syntax elements bit data after the encoded Slice Data (PAK Object Command) of the current Slice and prior to the next encoded Slice Data of the next Slice or prior to the end of the bistream, whichever comes first Cabac_Zero_Word or Trailing_Zero_8bits (as many bytes as there is).
Anything listed above before a Slice Data Context switch interrupt is not supported by this command.

DWord	Bit	Description

MFX_PAK_INSERT_OBJECT

0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_PAK_INSERT_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: [1h, FFFh] Excludes DWord (0,1) = Variable Length in DW Format: =n
1	31:18	**Reserved** Access: RO Format: MBZ
	17:16	**DataByteOffset - SrcDataStartingByteOffset[1:0]** Source Data Starting Byte Position within the very first inline DW. **Programming Notes** Must be set to 0 for JPEG encoder
	15	**HeaderLengthExcludeFrmSize** In case this flag is on, bits are NOT accumulated during current access unit coding neither for Cabac Zero Word insertion bits counting or for output in MMIO register MFC_BITSTREAM_BYTECOUNT_FRAME_NO_HEADER. When using HeaderLenghtExcludeFrmSize for header insertion, the software needs to make sure that data comes already with inserted start code emulation bytes. SW shouldn't set EmulationFlag bit (Bit 3 of DWORD1 of MFX_PAK_INSERT_OBJECT). **Value** | **Name** | **Description** 1 | NO_ACCUMULATION | Bits during current call are not accumulated 0 | ACCUMULATE | All bits accumulated

MFX_PAK_INSERT_OBJECT

Programming Notes
Must be set to 0 for JPEG encoder

14

Slice Header Indicator

Value	Name	Description
1	SLICE_HEADER	Insertion Object is a Slice Header. The command is stored internally by HW and is used for inserting slice headers.
0	LEGACY	Legacy Insertion Object command. The PAK Insertion Object command is not stored in HW.

13:8

DataBitsInLastDW - SrCDataEndingBitInclusion[5:0]

Source Data to be included in the very last inline DW. Follows the MSBit is the upper bit of each byte within the DW. The lower byte is actually processed first. For example, SrCDataEndingBitInclusion = 9, bit 7:0 and bit 15 are included as valid header data.

Value	Name
[1,32]	

7:4

SkipEmulByteCnt - Skip Emulation Byte Count

Skip emulation check for number of starting bytes It can be programmed from 0 to 15 bytes. For example, to skip the start code that has already prefixed in the bitstream.

Programming Notes
Must be set to 0 for JPEG encoder

3

EmulationFlag - EmulationByteBitsInsertEnable

Value	Name	Description
0	NONE	No emulation
1	EMULATE	Instruct the hardware to perform Start Code Prefix (0x 00 00 01/02/03/00) Search and Prevention Byte (0x 03) insertion on the insertion data of this command. It is required that hardware will handle a start code prefix crossing the boundary between insertion commands, or an insertion command followed by a PAK Object command.

Programming Notes
Must be set to 0 for JPEG encoder

2

LastHeaderFlag - LastSrcHeaderDataInsertCommandFlag

To process a series of consecutive insertion commands, this flag (=1) indicates the current command is the last 'header' insertion in the series. In CABAC, hardware must perform the "1" insert for byte align for Slice Header before Slice Data comes in in the next PAK-OBJECT command.In CAVLC, hardware ignores this bit

1

EndOfSliceFlag - LastDstDataInsertCommandFlag

No more insertion command and no more PAK-OBJECT command follows. Flush data out to memory

<table>
<tr><th colspan="3">MFX_PAK_INSERT_OBJECT</th></tr>
<tr><td></td><td>0</td><td><b>BitstreamStartReset - ResetBitStreamStartingPos</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td>RESET</td><td>Reset the bitstream buffer insertion position to the bitstream buffer starting position.</td></tr>
<tr><td>0</td><td>INSERT</td><td>Insert the current command inline data starting at the current bitstream buffer insertion position</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Must be set to 1 for JPEG encoder</td></tr>
</table></td></tr>
<tr><td>2..n</td><td>31:0</td><td><b>Insert Data PayLoad</b>
Actual Data to be inserted to the output bitstream buffer.</td></tr>
</table>

MFX_PIPE_BUF_ADDR_STATE

<table>
<tr><th colspan="3">MFX_PIPE_BUF_ADDR_STATE</th></tr>
<tr><td colspan="3">Source: VideoCS
Length Bias: 2</td></tr>
<tr><td colspan="3">This state command provides the memory base addresses for all row stores, StreamOut buffer and reconstructed picture output buffers required by the MFD or MFC Engine (that are in addition to the row stores of the Bit Stream Decoding/Encoding Unit (BSD/BSE) and the reference picture buffers).
This is a picture level state command and is common among all codec standards and for both encoder and decoder operating modes. However, some fields may only applicable to a specific codec standard. All Pixel Surfaces (original, reference frame and reconstructed frame) in the Encoder are programmed with the same surface state (NV12 and TileY format), except each has its own frame buffer base address. In the tile format, there is no need to provide buffer offset for each slice; since from each MB address, the hardware can calculated the corresponding memory location within the frame buffer directly.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline</b><table><tr><td>Default Value:</td><td>2h MFX_PIPE_BUF_ADDR_STATE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><b>Common Opcode</b><table><tr><td>Default Value:</td><td>0h MFX_COMMON_STATE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>23:21</td><td><b>SubOpcode A</b><table><tr><td>Default Value:</td><td>0h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>20:16</td><td><b>SubOpcode B</b><table><tr><td>Default Value:</td><td>2h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>Fixed Length<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>3Fh</td><td>DWORD_COUNT_n <b>[Default]</b></td><td>Excludes DWord (0,1)</td></tr></table></td></tr>
</table>

MFX_PIPE_BUF_ADDR_STATE		
1	31:6	**Pre Deblocking Destination Address** Format: GraphicsAddress[31:6] Specifies the 4K byte aligned frame buffer address for outputting the non-filtered reconstructed YUV picture (i.e. output of final adder in each codec standard, and prior to the deblocking filter unit). This field is ignored if PreDeblockOutEnable is set to 0 (disable).
	5:0	**Reserved** Access: RO Format: MBZ
2	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Pre Deblocking Destination Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Pre-Deblocking Destination Address. This field is ignored if **PreDeblockOutEnable** is set to 0 (disable).
3	31:15	**Reserved** Access: RO Format: MBZ
	14:13	**Reserved** Access: RO Format: MBZ
	12:11	**Reserved** Access: RO Format: MBZ
	10	**Compression Type** This field is valid only is Memory Compression is enabled. Value | Name 0 | Media Compression Enabled **[Default]** 1 | Render Compression Enabled
	9	**Pre Deblocking - Memory Compression Enable** Format: Enable Memory compression will be attempted for this surface. Value | Name 0 | Compression Disable 1 | Compression Enable

MFX_PIPE_BUF_ADDR_STATE

DWord	Bit	Description
		Programming Notes

Video Mode	Compression Enable
AVC Frame Only (No MBAFF or Field)	Yes
VP8 (Only Frame is supported	Yes

JPEG Decode

Chroma Format	Output Format	Compression Enable
422H_2Y,422H_4Y	YUY2	Yes
422H_2Y,422H_4Y	YUY2	Yes
422H_2Y,422H_4Y	UYVY	Yes
422H_2Y, 422H_4Y, 422V_2Y, 422V_4Y	NV12	No
420	YUY2, UYVY	No
420	NV12	Yes

DWord	Bit	Description
	8:7	**Pre Deblocking - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

DWord	Bit	Description
	6:0	**Pre Deblocking - Memory Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).
4	31:6	**Post Deblocking Destination Address** Format: GraphicsAddress[31:6] Specifies the 4K byte aligned frame buffer address for outputting the post-loop filtered reconstructed YUV picture (i.e. output of the deblocking filter unit) This field is ignored if PostDeblockOutEnable is set to 0 (disable).
	5:0	**Reserved** Access: RO Format: MBZ
5	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Post Deblocking Destination Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Post-Deblocking Destination Address. This field is ignored if **PostDeblockOutEnable** is set to 0 (disable).

MFX_PIPE_BUF_ADDR_STATE		
6	31:15	**Reserved** Access: RO Format: MBZ
	14:13	**Reserved** Access: RO Format: MBZ
	12:11	**Reserved** Access: RO Format: MBZ
	10	**Compression Type** This field is applicable only when Memory compression is enabled. Value | Name 0 | Media Compression Enabled **[Default]** 1 | Render Compression Enabled
	9	**Post Deblocking - Memory Compression Enable** Format: Enable Memory compression will be attempted for this surface. Value | Name 0 | Compression Disable 1 | Compression Enable
	8:7	**Post Deblocking - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value | Name 00b | Highest priority 01b | Second highest priority 10b | Third highest priority 11b | Lowest priority
	6:0	**Post Deblocking - Memory Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).
7	31:6	**Original Uncompressed Picture Source Address** Format: GraphicsAddress[31:6] Specifies the 64 byte aligned frame buffer address for fetching YUV pixel data from the original uncompressed input picture for encoding. This field is only valid in **encoding** mode.
	5:0	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">MFX_PIPE_BUF_ADDR_STATE</th></tr>
<tr><td>8</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Original Uncompressed Picture Source Address High</b><table><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table>This field is for the upper range of Original Uncompressed Picture Source Address. This field is valid for <b>encoding</b> mode only.</td></tr>
<tr><td>9</td><td>31:15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:13</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>12:11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Compression Type</b><table><tr><th>Description</th></tr><tr><td>This field is valid only when memory compression enable is true.</td></tr></table><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media Compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>Original Uncompressed Picture - Memory Compression Enable</b>
Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Original Uncompressed Picture Source - Arbitration Priority Control</b>
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>Highest priority</td></tr><tr><td>01b</td><td>Second highest priority</td></tr><tr><td>10b</td><td>Third highest priority</td></tr><tr><td>11b</td><td>Lowest priority</td></tr></table></td></tr>
</table>

MFX_PIPE_BUF_ADDR_STATE

	6:0	**Original Uncompressed Picture Source - Memory Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).
10	31:6	**StreamOut Data Destination Base Address** Format: GraphicsAddress[31:6] Specifies the 64 byte aligned address for outputting the per-MB indirect data to memory when **StreamOutEnable** is set in the MFX_PIPE_MODE_SELECT command. For Decoder: This field is used for transcoding purpose. For Encoder : This field is used for dynamic repeat of frame in PAK for Rate Control. Also used for feeding coding information back to the Host, Video Preprocessing Unit and ENC Unit. All data are written in fixed formats, and therefore all record sizes are known in the hardware. Hardware can calculate the offset into this base address for per-MB data.
	5:0	**Reserved** Access: RO Format: MBZ
11	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**StreamOut Data Destination Base Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Original Uncompressed Picture Source Address
12	31:15	**Reserved** Access: RO Format: MBZ
	14:13	**Reserved** Access: RO Format: MBZ
	12:10	**Reserved** Access: RO Format: MBZ
	9	**StreamOut Data Destination - Memory Compression Enable** Note: This is a READ Surface. The setting of this bit should match the settings on how this is written out before. **Value** / **Name** 0 / Compression Disable

<table>
<tr><th colspan="3">MFX_PIPE_BUF_ADDR_STATE</th></tr>
<tr><td></td><td>8:7</td><td><b>StreamOut Data Destination - Arbitration Priority Control</b>
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>Highest priority</td></tr>
<tr><td>01b</td><td>Second highest priority</td></tr>
<tr><td>10b</td><td>Third highest priority</td></tr>
<tr><td>11b</td><td>Lowest priority</td></tr>
</table></td></tr>
<tr><td></td><td>6:0</td><td><b>StreamOut Data Destination - Memory Object Control State</b>
<table>
<tr><td>Format:</td><td>MEMORY_OBJECT_CONTROL_STATE</td></tr>
</table>
Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).</td></tr>
<tr><td>13</td><td>31:6</td><td><b>Intra Row Store Scratch Buffer Base Address</b>
<table>
<tr><td>Format:</td><td>GraphicsAddress[31:6]</td></tr>
</table>
This field provides the base address of the scratch buffer (read/write) used by the AVC/VP8 IntraPrediction unit to store MB information of the previous row for processing of each macroblock in the current row. The Intra Row Store buffer must be 64-byte cacheline aligned. Hardware uses the horizontal address of the current macroblock to address the Intra Row Store. This field is ignored in MPEG2 and VC1 mode. Max 256 cachelines for 4K pixels (1 cacheline for either MBAFF or non-MBAFF)Intra Row Store Scratch Buffer - Arbitration Priority Control
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This is one of the four RowStore Scratch Buffers which can be programmed to use the internal Media Cache (total size 640 CacheLine). When Intra Row Store Scratch Buffer Cache Select is programmed to "1", this data will be stored inside MFX Media Internal Storage. Driver then needs to program this Base Address between 0 to 639, indicating starting cachelines address to Media Cache. Driver needs to make sure the whole buffer fits into MFX Media Internal Storage.
<i>(Notes: 1 cacheline per MB, and the buffer needs to have enough space for 1 MB row).</i></td></tr>
</table></td></tr>
<tr><td></td><td>5:0</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>14</td><td>31:16</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>15:0</td><td><b>Intra Row Store Scratch Buffer Base Address High</b>
<table>
<tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr>
</table>
This field is for the upper range of Intra RowStore/Scratch Buffer Base Address This field is ignored in MPEG2 and VC1 mode.</td></tr>
<tr><td>15</td><td>31:15</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">MFX_PIPE_BUF_ADDR_STATE</th></tr>
<tr><td></td><td>14:13</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>12</td><td><b>Intra Row Store Scratch Buffer Cache Select</b>
This field controls if Intra Row Store is going to store inside Media Cache or to LLC.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td></td><td>Buffer going to LLC.</td></tr><tr><td>1</td><td></td><td>Buffer going to Internal Media Storage</td></tr></table></td></tr>
<tr><td></td><td>11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Reserved - Intra Row Store</b></td></tr>
<tr><td></td><td>9</td><td><b>Intra Row Store Scratch Buffer - Memory Compression Enable</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>This surface is linear surface. This bit must be set to "0" since only TileY/TileYf/TileYs surface is allowed to be compressed</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Intra Row Store Scratch Buffer - Arbitration Priority Control</b>
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>Highest priority</td></tr><tr><td>01b</td><td>Second highest priority</td></tr><tr><td>10b</td><td>Third highest priority</td></tr><tr><td>11b</td><td>Lowest priority</td></tr></table></td></tr>
<tr><td></td><td>6:0</td><td><b>Intra Row Store Scratch Buffer - Memory Object Control State</b><table><tr><td>Format:</td><td>MEMORY_OBJECT_CONTROL_STATE</td></tr></table>Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).</td></tr>
<tr><td>16</td><td>31:6</td><td><b>Deblocking Filter Row Store Scratch Base Address</b><table><tr><td>Format:</td><td>GraphicsAddress[31:6]</td></tr></table>Deblocking Filter Row Store is needed for:<ul><li>AVC and VC1 In-Loop Deblocking Filter</li><li>VC1 Overlap-smoothing Filter</li><li>AVC, VC1, and MPEG-2 Out-Of-Loop Deblocking Filter (Intel extension)</li></ul>This field provides the 64 byte aligned base address of the scratch buffer (read and write) used by the deblocking filter unit to store MB information of the previous row for filtering of each</td></tr>
</table>

MFX_PIPE_BUF_ADDR_STATE

<table>
<tr><td></td><td></td><td>macroblock in the current row. The Deblocking Filter Row Store buffer must be 64-byte cacheline aligned. Hardware uses the horizontal address of the current macroblock to address the Deblocking Filter Row Store. Max 6 cachelines for VC1 and MPEG2, and max 4 for AVC (for MBAFF, 2 for non-MBAFF)
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This is one of the four RowStore Scratch Buffers which can programmed to use the internal Media Cache (total size 640 CacheLine). When Deblocking Filter Row Store Scratch Buffer Cache Select is programmed to "1", this will be stored inside MFX Media Internal Storage. Driver then needs to program this Base Address between 0 to 639, indicating starting cachelines address location for this buffer. Driver needs to make sure the whole buffer fits into Media Internal Storage.
<i>(Notes: 2 cachelines per MB for non-mbaff; 4 cachelines per MB pair for mbaff, and the buffer needs to have enough space for 1 MB (pair) row).</i></td></tr>
</table></td></tr>
<tr><td></td><td>5:0</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>17</td><td>31:16</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>15:0</td><td><b>Deblocking Filter Row Store Scratch Base Address High</b>
<table>
<tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr>
</table>
This field is for the upper range of Deblocking Filter Row Store Scratch Buffer Address.</td></tr>
<tr><td>18</td><td>31:15</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>14:13</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>12</td><td><b>Deblocking Filter Row Store Scratch Buffer Cache Select</b>
This field controls if Intra Row Store is going to store inside Media Internal Storage or to LLC.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td></td><td>Buffer going to LLC</td></tr>
<tr><td>1</td><td></td><td>Buffer going to Media Internal Storage</td></tr>
</table></td></tr>
<tr><td></td><td>11</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>10</td><td><b>Deblocking Filter Row Store Scratch - Memory Compression Mode</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Reserved <b>[Default]</b></td></tr>
</table></td></tr>
</table>

MFX_PIPE_BUF_ADDR_STATE

	9	**Deblocking Filter Row Store Scratch - Memory Compression Enable**
		Value | **Name** 0 | Compression Disable
		Programming Notes This surface is linear surface. This bit must be set to "0" since only TileY/TileYf/TileYs surface is allowed to be compressed
	8:7	**Deblocking Filter Row Store Scratch - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.
		Value | **Name** 00b | Highest priority 01b | Second highest priority 10b | Third highest priority 11b | Lowest priority
	6:0	**Deblocking Filter Row Store Scratch - Memory Object Control State**
		Format: | MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).
19..50	1023:0	**Reference Picture Base Addr**
		Format: | MFX_REFERENCE_PICTURE_BASE_ADDR[16]
51	31:15	**Reserved**
		Access: | RO Format: | MBZ
	14:13	**Reserved**
		Access: | RO Format: | MBZ
	12:9	**Reserved**
		Access: | RO Format: | MBZ
	8:7	**Reference Picture - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.
		Value | **Name** 00b | Highest priority 01b | Second highest priority 10b | Third highest priority 11b | Lowest priority

MFX_PIPE_BUF_ADDR_STATE

	6:0	**Reference Picture - Memory Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).
52	31:6	**Macroblock Buffer Base Address or Decoded Picture Error/Status Buffer Base Address** Format: GraphicsAddress[31:6] **For decoder:** Specifies the 64 byte aligned buffer address for writing a single error/status record into the memory when **Pic Error/Status Report Enable** is set in the MFX_PIPE_MODE_SELECT Command. The error/status record is written by HW at the end of decoding one single picture. The record is written in a fixed format, total 96-bits in size always. Please refer to "Media VDBOX -> Video Codec -> Other Codec Functions -> MFX Error Handling -> Decoder" session for the output format. **For encoder:** Specifies the 64 byte aligned buffer address for reading the per-MB indirect data from memory when **MacroblockStatEnable** is set in the MFX_AVC_IMG_STATE Command. This field is used for dynamic repeat of frame in PAK for Rate Control. Also used for feeding coding information back to the Host, Video Preprocessing Unit, and ENC Unit. All data are written in fixed formats, and therefore all record sizes are known in the hardware. Hardware can calculate the offset into this base address for per-MB data.
	5:0	**Reserved** Access: RO Format: MBZ
53	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Macroblock Buffer Base Address or Decoded Picture Error/Status Buffer Base Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Macroblock Status Buffer Base Address
54	31:15	**Reserved** Access: RO Format: MBZ
	14:13	**Reserved** Access: RO Format: MBZ
	12:11	**Reserved** Access: RO Format: MBZ
	10	**Macroblock Status Buffer - Memory Compression Mode** Value: Name 0: Reserved **[Default]**

MFX_PIPE_BUF_ADDR_STATE

DWord	Bit	Description
	9	**Macroblock Status Buffer - Memory Compression Enable** Value: 0 — Name: Compression Disable **Programming Notes** This surface is linear surface. This bit must be set to "0" since only TileY/TileYf/TileYs surface is allowed to be compressed
	8:7	**Macroblock Status Buffer - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value: 00b — Name: Highest priority Value: 01b — Name: Second highest priority Value: 10b — Name: Third highest priority Value: 11b — Name: Lowest priority
	6:0	**Macroblock Status Buffer - Memory Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).
55	31:6	**Macroblock ILDB StreamOut Buffer Base Address** Format: GraphicsAddress[31:6] Specifies the 64 byte aligned buffer address for writing MB ILDB parameter per MB to memory when **Debocker streamout enable** is set in the MFX_PIPE_MODE_SELECT Command. The ildb MB control parameters are written by HW at the end of each decoding MB. Only AVC edge information is being streamed out. It is used in AVC decode mode only.
	5:0	**Reserved** Access: RO Format: MBZ
56	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Macroblock ILDB StreamOut Buffer Base Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Deblocking Filter Row Store Scratch Address
57	31:15	**Reserved** Access: RO Format: MBZ

MFX_PIPE_BUF_ADDR_STATE

<table>
<tr><td></td><td>14:13</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>12:11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Macroblock ILDB StreamOut Buffer - Memory Compression Mode</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Reserved <b>[Default]</b></td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>Macroblock ILDB StreamOut Buffer - Memory Compression Enable</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>This surface is linear surface. This bit must be set to "0" since only TileY/TileYf/TileYs surface is allowed to be compressed</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Macroblock ILDB StreamOut Buffer - Arbitration Priority Control</b>
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>Highest priority</td></tr><tr><td>01b</td><td>Second highest priority</td></tr><tr><td>10b</td><td>Third highest priority</td></tr><tr><td>11b</td><td>Lowest priority</td></tr></table></td></tr>
<tr><td></td><td>6:0</td><td><b>Macroblock ILDB StreamOut Buffer - Memory Object Control State</b><table><tr><td>Format:</td><td>MEMORY_OBJECT_CONTROL_STATE</td></tr></table>Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).</td></tr>
<tr><td>58</td><td>31:6</td><td><b>Second Macroblock ILDB StreamOut Buffer Base Address</b><table><tr><td>Format:</td><td>GraphicsAddress[31:6]</td></tr></table>64 byte aligned buffer. Specifies the 64 byte aligned buffer address for writing MB ILDB parameter per MB to memory when Debocker streamout enable is set in the MFX_PIPE_MODE_SELECT Command. The ildb MB control parameters are written by HW at the end of each decoding MB. Only AVC edge information is being streamed out. It is used in AVC decode mode only.</td></tr>
<tr><td></td><td>5:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFX_PIPE_BUF_ADDR_STATE</th></tr>
<tr><td>59</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Second Macroblock ILDB StreamOut Buffer Base Address High</b><table><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table>This field is for the upper range of Second Macroblock ILDB StreamOutBuffer Base Address.</td></tr>
<tr><td>60</td><td>31:15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:13</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>12:11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Second Macroblock ILDB StreamOut Buffer - Memory Compression Mode</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Reserved <b>[Default]</b></td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>Second Macroblock ILDB StreamOut Buffer - Memory Compression Enable</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>This surface is linear surface. This bit must be set to "0" since only TileY/TileYf/TileYs surface is allowed to be compressed</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Second Macroblock ILDB StreamOut Buffer - Arbitration Priority Control</b>
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.</td></tr>
<tr><td></td><td>6:0</td><td><b>Second Macroblock ILDB StreamOut Buffer - Memory Object Control State</b><table><tr><td>Format:</td><td>MEMORY_OBJECT_CONTROL_STATE</td></tr></table>Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).</td></tr>
<tr><td>61</td><td>31</td><td><b>Reference Picture 15 - Compression Type</b>
This field is valid only when memory compression is enabled.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media Compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFX_PIPE_BUF_ADDR_STATE</th></tr>
<tr><td></td><td>30</td><td><b>Reference Picture 15 - Memory Compression Enable</b>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr><tr><td>1</td><td>Compression Enable</td></tr></table></td></tr>
<tr><td></td><td>29</td><td><b>Reference Picture 14 - Compression Type</b>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media Compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>Reference Picture 14 - Memory Compression Enable</b>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr><tr><td>1</td><td>Compression Enable</td></tr></table></td></tr>
<tr><td></td><td>27</td><td><b>Reference Picture 13 - Compression Type</b>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media Compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>26</td><td><b>Reference Picture 13 - Memory Compression Enable</b>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr><tr><td>1</td><td>Compression Enable</td></tr></table></td></tr>
<tr><td></td><td>25</td><td><b>Reference Picture 12 - Compression Type</b>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media Compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>24</td><td><b>Reference Picture 12 - Memory Compression Enable</b>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr><tr><td>1</td><td>Compression Enable</td></tr></table></td></tr>
<tr><td></td><td>23</td><td><b>Reference Picture 11 - Compression Type</b>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media Compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
</table>

MFX_PIPE_BUF_ADDR_STATE

22 **Reference Picture 11 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

21 **Reference Picture 10-Compression Type**

Value	Name
0	Media Compression Enabled **[Default]**
1	Render compression enabled

20 **Reference Picture 10 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

19 **Reference Picture 9 - Compression Type**

Value	Name
0	Media Compression Enabled **[Default]**
1	Render Compression Enabled

18 **Reference Picture 9 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

17 **Reference 8 - Compression Type**

Value	Name
0	Media Compression Enabled **[Default]**
1	Render Compression Enabled

16 **Reference Picture 8 - Memory Compression Enable**

Value	Name
0	Compression Disable
1	Compression Enable

15 **Reference Picture 7 - Compression Type**

Value	Name
0	Media Compression Enabled **[Default]**
1	Render Compression Enabled

<table>
<tr><th colspan="3">MFX_PIPE_BUF_ADDR_STATE</th></tr>
<tr><td></td><td>14</td><td><b>Reference Picture 7 - Memory Compression Enable</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr><tr><td>1</td><td>Compression Enable</td></tr></table></td></tr>
<tr><td></td><td>13</td><td><b>Reference Picture 6 - Compression Type</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media Compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>12</td><td><b>Reference Picture 6 - Memory Compression Enable</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr><tr><td>1</td><td>Compression Enable</td></tr></table></td></tr>
<tr><td></td><td>11</td><td><b>Reference Picture 5 -Compression Type</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media Compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Reference Picture 5 - Memory Compression Enable</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr><tr><td>1</td><td>Compression Enable</td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>Reference PIcture 4 - Compression Type</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media Compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>8</td><td><b>Reference Picture 4 - Memory Compression Enable</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr><tr><td>1</td><td>Compression Enable</td></tr></table></td></tr>
<tr><td></td><td>7</td><td><b>Reference PIcture 3 - Compression Type</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media Compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFX_PIPE_BUF_ADDR_STATE</th></tr>
<tr><td></td><td>6</td><td><b>Reference Picture 3 - Memory Compression Enable</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr><tr><td>1</td><td>Compression Enable</td></tr></table></td></tr>
<tr><td></td><td>5</td><td><b>Reference Picture 2 - Compression Type</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media Compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>4</td><td><b>Reference Picture 2 - Memory Compression Enable</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr><tr><td>1</td><td>Compression Enable</td></tr></table></td></tr>
<tr><td></td><td>3</td><td><b>Reference Picture 1 - Compression Type</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media Compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>2</td><td><b>Reference Picture 1 - Memory Compression Enable</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr><tr><td>1</td><td>Compression Enable</td></tr></table></td></tr>
<tr><td></td><td>1</td><td><b>Reference Picture 0 - Compression Type</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Media Compression Enabled <b>[Default]</b></td></tr><tr><td>1</td><td>Render Compression Enabled</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>Reference Picture 0 - Memory Compression Enable</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Compression Disable</td></tr><tr><td>1</td><td>Compression Enable</td></tr></table></td></tr>
<tr><td>62</td><td>31:6</td><td><b>Scaled Reference Surface Base Address</b><table><tr><td>Format:</td><td>GraphicsAddress[31:6]</td></tr></table>Specifies the 64 byte aligned down scaled reference frame buffer addresses that needs to be used by the PAK down-scaler to write the down scaled pixels. Only the luma pixels will be downscaled and written to the surface</td></tr>
</table>

MFX_PIPE_BUF_ADDR_STATE

<table>
<tr><td></td><td>5:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>63</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Scaled Reference Surface Base Address High</b><table><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table>This field is for the upper range of Scaled Reference Surface Base Address.</td></tr>
<tr><td>64</td><td>31:15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:13</td><td><b>Reserved14_13</b></td></tr>
<tr><td></td><td>12:11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Scaled Reference Surface - Render Compression Enable</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Disable <b>[Default]</b></td></tr><tr><td>1</td><td>Enable</td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>Scaled Reference Surface - Memory Compression Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>Memory compression shouldn't be enabled for this surface.</td></tr>
<tr><td></td><td>8:7</td><td><b>Scale Reference Surface - Arbitration Priority Control</b>
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>Highest Priority</td></tr><tr><td>01b</td><td>Second Highest Priority</td></tr><tr><td>10b</td><td>Third Highest Priority</td></tr><tr><td>11b</td><td>Lowest Priority</td></tr></table></td></tr>
<tr><td></td><td>6:0</td><td><b>Scaled Reference Surface - Memory Object Control State</b><table><tr><td>Format:</td><td>MEMORY_OBJECT_CONTROL_STATE</td></tr></table>Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).</td></tr>
</table>

<table>
<tr><th colspan="3">MFX_PIPE_BUF_ADDR_STATE</th></tr>
<tr><td>65</td><td>31:6</td><td><b>SliceSize StreamOut Data Destination Base Address</b><table><tr><td>Format:</td><td>GraphicsAddress[31:6]</td></tr></table>Specifies the 64 byte aligned Slice Size streamout surface address. Here slice sizes are written out. This surface can be used to determine the slice start location.</td></tr>
<tr><td></td><td>5:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>66</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>SliceSize StreamOut Data Destination Base Address High</b><table><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table>This field is for the upper range of Slice Size Streamout Surface Base Address.</td></tr>
<tr><td>67</td><td>31:15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:13</td><td><b>Reserved14_13</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td></td></tr></table></td></tr>
<tr><td></td><td>12:11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>SliceSize StreamOut Data Destination - Memory Compression Mode</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Reserved <b>[Default]</b></td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>SliceSize StreamOut Data Destination - Memory Compression Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>Memory compression is never enabled for this surface</td></tr>
<tr><td></td><td>8:7</td><td><b>SliceSize StreamOut Data Destination - Arbitration Priority Control</b>
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>Highest Priority</td></tr><tr><td>01b</td><td>Second Highest Priority</td></tr><tr><td>10b</td><td>Third Highest Priority</td></tr><tr><td>11b</td><td>Lowest Priority</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFX_PIPE_BUF_ADDR_STATE</th></tr>
<tr><td></td><td>6:0</td><td><b>SliceSize StreamOut Data Destination - Memory Object Control State</b><table><tr><td>Format:</td><td>MEMORY_OBJECT_CONTROL_STATE</td></tr></table>Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).</td></tr>
</table>

MFX_PIPE_MODE_SELECT

MFX_PIPE_MODE_SELECT

Source:	VideoCS
Length Bias:	2

Specifies which codec and hardware module is being used to encode/decode the video data, on a per-frame basis.
The MFX_PIPE_MODE_SELECT command specifies which codec and hardware module is being used to encode/decode the video data, on a per-frame basis. It also configures the hardware pipeline according to the active encoder/decoder operating mode for encoding/decoding the current picture. Commands issued specifically for AVC and MPEG2 are ignored when VC1 is the active codec.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_COMMON Format: OpCode
	26:24	**Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpA** Default Value: 0h Format: OpCode
	20:16	**SubOpB** Default Value: 0h MFX_PIPE_MODE_SELECT Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Format: =n Value | Name | Description 3h | DWORD_COUNT_n **[Default]** | Excludes DWord (0,1)
1	31:24	**AES Control** Format: AES_CONTROL

MFX_PIPE_MODE_SELECT

23:19 **Reserved**

Access:	RO
Format:	MBZ

18 **Reserved**

17 **Decoder Short Format Mode**

For IT mode, this bit must be 0.

Value	Name	Description
0	Short Format Driver Interface **[Default]**	AVC/VC1/MVC/VP8 Short Format Mode is in use **Note: There is no Short Format for VP8 yet, so this field must be set to 1 for VP8.**
1	Long Format Driver Interface	AVC/VC1/MVC/VP8 Long Format Mode is in use.

16:15 **Decoder Mode select**

Each coding standard supports two entry points: VLD entry point and IT (IDCT) entry point. This field selects which one is in use. This field is only valid if Codec Select is 0 (decoder).

Value	Name	Description
0h	VLD Mode	All codec minimum must support this mode Configure the MFD Engine for VLD Mode Note: All codec minimum must support this mode
1h	IT Mode	Configure the MFD Engine for IT Mode Note: Only VC1 and MPEG2 support this mode
2h	Deblocker Mode	Configure the MFD Engine for Standalone Deblocker Mode. Require streamout AVC edge control information from preceding decoding pass.
3h	Interlayer Mode	Configure the MFX Engine for standalone interlayer upsampling for motion info, residual and reconstructed pixel. Require information being streamout from the preceding encoding and decoding pass of a reference layer.>

14 **Reserved**

13 **Reserved**

12 **Deblocker Stream-Out Enable**

This field indicates if Deblocker information is going to be streamout during VLD decoding. For AVC, it is needed to enable the deblocker streamout as the AVC Disable_DLKFilterIdc is a slice level parameters. Driver needs to determine ahead of time if at least one slice of the current frame/ has deblocker ON.

Value	Name	Description
0h	Disable	Disable streamout of deblocking control information for standalone deblocker operation. It needs other fields to determine one or two deblocking surface streamout (Post Deblocking Output Enable, Pre Deblocking Output Enable, interlayer idc and regular deblock idc).
1h	Enable	

<table>
<tr><th colspan="3">MFX_PIPE_MODE_SELECT</th></tr>
<tr><td></td><td>11</td><td><b>Pic Error/Status Report Enable.</b>
This field control whether the error/status reporting is enable or not.0: Disable1: EnableIn decoder modes: Error reporting is written out once per frame. The Error Report frame ID listed in DW3 along with the VLD/IT error status bits are packed into one cache and written to the "Decoded Picture Error/Status Buffer address" listed in the MFX_PIPE_BUF_ADDR_STATE Command. Note: driver shall program different error buffer addresses between pictures; otherwise, hardware might overwrite previous written data if driver does not read it fast enough. In encoder modes: Not used
Please refer to "Media VDBOX -> Video Codec -> Other Codec Functions -> MFX Error Handling -> Decoder" session for the output format.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0h</td><td>Disable</td></tr>
<tr><td>1h</td><td>Enable</td></tr>
</table></td></tr>
<tr><td></td><td>10</td><td><b>Stream-Out Enable</b>
This field controls whether the macroblock parameter stream-out is enabled during VLD decoding for transcoding purpose.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0h</td><td>Disable</td></tr>
<tr><td>1h</td><td>Enable</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>In decoder modes: The Stream-Out feature is added to support transcoding. While decoding the input compressed stream, selected decoded information may be used by the encoder for re-compression. In encoder modes: This feature used to perform dynamic Multipass of PAK for conformance purpose. Also it provides feedback to host (ENC) for future needs. Software can use this bit to disable writing PAK steam data to the streamout buffer for last pass of frame in PAK. Thus, save memory bandwidth.</td></tr>
</table></td></tr>
<tr><td></td><td>9</td><td><b>Post Deblocking Output Enable (PostDeblockOutEnable)</b>
This field controls the output write for the reconstructed pixels AFTER the deblocking filter. In MPEG2 decoding mode, if this is enabled, VC1 deblocking filter is used.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0h</td><td>Disable</td></tr>
<tr><td>1h</td><td>Enable</td></tr>
</table></td></tr>
<tr><td></td><td>8</td><td><b>Pre Deblocking Output Enable (PreDeblockOutEnable)</b>
This field controls the output write for the reconstructed pixels BEFORE the deblocking filter.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0h</td><td>Disable</td></tr>
<tr><td>1h</td><td>Enable</td></tr>
</table></td></tr>
<tr><td></td><td>7</td><td><b>Scaled Surface Enable</b>
This field indicates if the scaled surface is enabled. This field enables the 4x HME downscalar of the reconstructed image. Only supported for AVC and VP8 formats.</td></tr>
</table>

<table>
<tr><th colspan="3">MFX_PIPE_MODE_SELECT</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>0h</td><td>Disable</td></tr><tr><td>1h</td><td>Enable</td></tr></table></td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>5</td><td><b>Stitch Mode</b><table><tr><td>Exists If:</td><td>//CodecSel=Encode and StandardSel=AVC</td></tr></table><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Not in stitch mode</td><td></td></tr><tr><td>1h</td><td>In the special stitch mode</td><td>This mode can be used for any Codec as long as bitfield conditions are met.</td></tr></table></td></tr>
<tr><td></td><td>4</td><td><b>Codec Select</b><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Decode</td><td></td></tr><tr><td>1h</td><td>Encode</td><td>Valid only if StandardSel is AVC and MPEG2)</td></tr></table></td></tr>
<tr><td></td><td>3:0</td><td><b>Standard Select</b><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0000b</td><td>MPEG2</td><td></td></tr><tr><td>0001b</td><td>VC1</td><td></td></tr><tr><td>0010b</td><td>AVC</td><td>Covers both AVC and MVC</td></tr><tr><td>0011b</td><td>JPEG</td><td></td></tr><tr><td>0101b</td><td>VP8</td><td>Decoder, Encoder</td></tr><tr><td>0110b</td><td>Reserved</td><td></td></tr><tr><td>0111b</td><td>Reserved</td><td></td></tr><tr><td>1111b</td><td>UVLD</td><td>SW decoder w/ embedded micro-controller and co-processor</td></tr></table></td></tr>
<tr><td>2</td><td>31:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MFX_PIPE_MODE_SELECT</th></tr>
<tr><td>3</td><td>31:0</td><td><b>Pic Status/Error Report ID</b>
<table>
<tr><td>Exists If:</td><td>//Decoder Mode Only</td></tr>
<tr><td>Format:</td><td>U32</td></tr>
</table>
In decoder modes: Error reporting is written out once per frame. This field along with the VLD error status bits are packed into one cache and written to the memory location specified by "Decoded Picture Error/Status Buffer address" listed in the MFX_PIPE_BUF_ADDR_STATE Command.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>32-bit unsigned</td><td>Unique ID Number</td></tr>
<tr><td>1h</td><td>Reserved</td><td></td></tr>
</table>
</td></tr>
<tr><td>4</td><td>31:0</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
</table>

MFX_QM_STATE

MFX_QM_STATE

Source:	VideoCS
Length Bias:	2

This is a common state command for AVC encoder modes. For encoder, it represents both the forward QM matrices as well as the decoding QM matrices. This is a Frame-level state. Only Scaling Lists specified by an application are being sent to the hardware. The driver is responsible for determining the final set of scaling lists to be used for decoding the current slice, based on the AVC Spec Table 7-2 (Fall-Back Rules A and B). In MFX AVC PAK mode, PAK needs both forward Q scaling lists and IQ scaling lists. The IQ scaling lists are sent as in MFD in raster scan order. But the Forward Q scaling lists are sent in column-wise raster order (column-by-column) to simplify the H/W. Driver will perform all the scan order conversion for both ForwardQ and IQ.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 7h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 20h Excludes DWord (0,1) Format: =n
1	31:2	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">MFX_QM_STATE</th></tr>
<tr><td></td><td>1:0</td><td><b>AVC</b>
<table><tr><td>Exists If:</td><td>//AVC- Decoder Only</td></tr></table>
<b>For AVC QM Type</b>: This field specifies which Quantizer Matrix is loaded.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>AVC_4x4_Intra_MATRIX, (Y-4DWs, Cb-4DWs, Cr-4DWs, reserved-4DWs)</td></tr>
<tr><td>1</td><td>AVC_4x4_Inter_MATRIX, (Y-4DWs, Cb-4DWs, Cr-4DWs, reserved-4DWs)</td></tr>
<tr><td>2</td><td>AVC_8x8_Intra_MATRIX</td></tr>
<tr><td>3</td><td>AVC_8x8_Inter_MATRIX</td></tr>
</table></td></tr>
<tr><td></td><td>1:0</td><td><b>MPEG2</b>
<table><tr><td>Exists If:</td><td>//MPEG2- Decoder Only</td></tr></table>
<b>For MPEG2 QM Type</b>: This field specifies which Quantizer Matrix is loaded.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>MPEG_INTRA_QUANTIZER_MATRIX</td></tr>
<tr><td>1</td><td>MPEG_NON_INTRA_QUANTIZER_MATRIX</td></tr>
<tr><td>2-3</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td></td><td>1:0</td><td><b>JPEG</b>
<table><tr><td>Exists If:</td><td>//JPEG- Encoder Only</td></tr></table>
<b>For JPEG QM Type</b>: This field specifies which Quantizer Matrix is loaded.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>JPEG_Luma_Y_QUANTIZER_MATRIX (or R)</td></tr>
<tr><td>1</td><td>JPEG_Chroma_Cb_QUANTIZER_MATRIX (or G)</td></tr>
<tr><td>2</td><td>JPEG_Chroma_Cr_QUANTIZER_MATRIX (or B)</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>For JPEG encoder, each quantization element presents 16-bit 1/QM[i][j].In RGB encoding, because the order input image components can be RGB, GBR, BGR, YUV, the value 0 is used for the first image component, the value 1 is used for the second image component, and the value 2 is used for the third image component.</td></tr>
</table></td></tr>
<tr><td>2..33</td><td>1023:0</td><td><b>Forward Quantizer Matrix</b>
The format of a Quantizer Matrix is an 8x8 matrix in raster order. Each element is an unsigned byte.</td></tr>
</table>

MFX_STATE_POINTER

MFX_STATE_POINTER

Source:	VideoCS
Length Bias:	2

The MFX_STATE_POINTER command, issued at picture level, is used to set up the indirect pointers for VCS to fetch all the MFX states (Image state, Slice state, etc.) needed for the encoding/decoding process in PAK/IT mode. The encoding/decoding states are presented by state commands, which are grouped into separate sets (picture level, slice level, etc.), and each is stored in its own memory buffer referred by an indirect state pointer. The content of each indirect state buffer is a list of MFX state commands with no special format requirements. The sequence of commands in each indirect state buffer is terminated by a MI_BATCH_BUFFER_END command(acts as the last command marker). Therefore, indirect state buffers can have different and variable length of command sequences.

The indirection is designed to facilitate context switching in the middle of a codec operation. The smallest granularity of interruption is designed to be at a completed MB row in AVC/VC1/MPEG2 IT and AVC PAK operating modes as well as in VC1/MPEG2 VLD mode. There is no support for context switch in AVC VLD mode.

Hardware supports up to 4 separate indirect state pointers, allowing software to manage the grouping of state commands. During context switch, hardware re stores (re-issues) the latest version of each indirect state pointer, if present.

MFX_STATE_POINTER command can only program one indirect state pointer at a time. MI_FLUSH will invalidate all indirect state buffer pointers inside VCS.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFX_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 6h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">MFX_STATE_POINTER</th></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b><table><tr><td>Default Value:</td><td>0h DWORD_COUNT_n</td></tr><tr><td>Format:</td><td>=n</td></tr></table></td></tr>
<tr><td>1</td><td>31:5</td><td><b>State Pointer</b><table><tr><td>Format:</td><td>GeneralStateOffset[31:5]</td></tr></table>Specifies the 32-byte aligned address of an Indirect State Buffer. This pointer is relative to the General State Base Address.</td></tr>
<tr><td></td><td>4:2</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>1:0</td><td><b>State Pointer Index</b>
Specifies one of the four indirect state pointers to program.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>00b</td><td></td><td>indirect state pointer 0 (image state)</td></tr><tr><td>01b</td><td></td><td>indirect state pointer 1 (slice state)sc</td></tr><tr><td>10b</td><td></td><td>indirect state pointer 2</td></tr><tr><td>11b</td><td></td><td>indirect state pointer 3</td></tr></table></td></tr>
</table>

MFX_STITCH_OBJECT

MFX_STITCH_OBJECT

Source: VideoCS
Length Bias: 2

The MFC_STITCH_OBJECT command is used when stitch-enabled is set to 1, while CodecSel and StandardSel are set to ENCODE and AVC, respectively. This command is used, for example, to stitch multiple bitstreams to form a transport stream.

It is a variable length command as the data to be inserted are presented as either inline data and/or indirect data of this command. Multiple insertion commands can be issued back-to-back in a series. It is host software's responsibility to make sure their corresponding data will properly stitch together to form a valid output.

Hardware keeps track of an output bitstream buffer current byte position and the associated next bit insertion position index. Context switch interrupt is not supported by this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_STITCH_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Format: OpCode
	20:16	**SubOpcode B** Default Value: Ah Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: [0h, FFFh] Excludes DWord (0,1) = Variable Length in DW (>= 3) Format: =n If it is 3, it indicates the absent of inline data.
1	31:18	**Reserved** Access: RO Format: MBZ

MFX_STITCH_OBJECT

	17:16	**Source Data Starting Byte Offset** Source Data Starting Byte Position within the very first inline DW.
	15:14	**Reserved** Access: RO Format: MBZ
	13:8	**Source Data Ending Bit Inclusion** Source Data to be included in the very last inline DW. Follows the MSBit is the upper bit of each byte within the DW. The lower byte is actually processed first. For example, SrCDataEndingBitInclusion =9, bit 7:0 and bit 15 are included as valid header data. **Value** / **Name** [1,32] /
	7:4	**Reserved**
	3	**Reserved**
	2	**Last Source Header Data Insert Command Flag** To process a series of consecutive insertion commands, this flag (=1) indicates the current command is the last 'header' insertion in the series. In CABAC, hardware must perform the "1" insert for byte align for Slice Header before Slice Data comes in in the next PAK-OBJECT command. In CAVLC, hardware ignores this bit.
	1	**Last Destination Data Insert Command Flag** THIS FIELD MUST BE THE SAME AS Last Source Header Data Insert Command Flag No more insertion command and no more PAK-OBJECT command follows. Flush data out to memory
	0	**Reserved**
2	31:19	**Reserved** Access: RO Format: MBZ
	18:0	**Indirect Data Length** Format: U19 This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address.
3	31:0	**Indirect Data Start Address** Format: GraphicsAddress[31:0] This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the MFX Indirect Bitstream Object Base Address. Hardware ignores this field if indirect data is not present.
4..n	31:0	**Insert Data PayLoad** Inline data to be inserted to the output bitstream buffer

MFX_SURFACE_STATE

MFX_SURFACE_STATE

Source:	VideoCS
Length Bias:	2

Description

This command is common for all encoding/decoding modes, tospecify the uncompressed YUV picture (i.e. destination surface) or intermediate streamout in/out surface (e.g. coefficient/residual) (field, frame or interleaved frame) format for reading and writing:

- Uncompressed, original input picture to be encoded
- Reconstructed non-filtered/filtered display picture (becoming reference pictures as well for subsequent temporal inter-prediction)

Since there is only one media surface state being active during the entire encoding/decoding process, all the uncompressed/reconstructed pictures are defined to have the same surface state. The primary difference among picture surface states is their individual programmed base addresses, which are provided by other state commands and not included in this command. MFX engine is making the association of surface states and corresponding buffer base addresses.
MFX engine currently supports only one media surface type for video and that is the NV12 (Planar YUV420 with interleaved U (Cb) and V (Cr). For optimizing memory efficiency based on access patterns, only TileY is supported. For JPEG decoder, only IMC1 and IMC3 are supported. Pitch can be wider than the Picture Width in pixels and garbage will be there at the end of each line. The following describes all the different formats that are supported and not supported in MFX :

- NV12 - 4:2:0 only; UV interleaved; Full Pitch, U and V offset is set to 0 (the only format supported for video codec); vertical UV offset is MB aligned; UV xoffsets = 0. JPEG does not supportNV12 format because non-interleave JPEG has performance issue with partial write (in interleaved UV format)
- IMC 1 & 3 - Full Pitch, U and V are separate plane; (JPEG only; U plane + garbage first in full pitch followed by V plane + garbage in full pitch). U and V vertical offsets are block aligned; U and V xoffset = 0; there is no gap between Y, U and V planes. IMC1 and IMC3 are different by a swap of U and V. This is the only format supported in JPEG for all video subsampling types (4:4:4, 4:2:2 and 4:2:0)
- We are not supporting IMC 2 & 4 - Full Pitch, U and V are separate plane (JPEG only; Uplane first in full pitch followed by V plane in full pitch - U and V plane are side-by-side). U and V vertical offsets are 16-pixel aligned; V xoffset is half-pitch aligned; U xoffset is 0; there is no gap between Y, U and V planes. IMC2 and IMC4 are different by a swap of U and V.
- We are not supporting YV12 - half pitch for each U and Vplane, and separate planes for Y, U and V (U plane first in half pitch followed by Vplane in half pitch). For YV12, U and V vertical offsets are block aligned; U and Vxoffset = 0; there is no gap between Y, U and V planes

Note that the following datastructures are not specified through the media surface state

- 1D buffers for row-store and other miscellaneous information.
- 2D buffers for per-MB data-structures (e.g. DMV biffer, MB info record, ILDB Control and Tcoeff/Stocoeff).

This surface state here is identical to the Surface State for deinterlace and sample_8x8messages described in the Shared Function Volume.

MFX_SURFACE_STATE

For non pixel data, such as row stores, indirect data (Compressed Slice Data, AVC MV record, Coeff record and AVC ILDB record) and streamin/out and output compressed bitstream, a linear buffer is employed. For row stores, the H/W is designed to guarantee legal memory accesses (read and write). For the remaining cases, indirect object base address, indirect object address upper bound, object data start address (offset) and object data length are used to fully specified their corresponding buffer. This mechanism is chosen over the pixel surface type because of their variable record sizes.
All row store surfaces are linear surface. Their addresses are programmed in Pipe_Buf_Base_State orBsp_Buf_Base_Addr_State

This surface state here is identical to the Surface State for deinterlace and sample_8x8messages described in the Shared Function Volume and Sampler Chapter.

Programming Notes

VC1 I picture scaling: Even though VC1 allows I reconstructed picture scaling (via RESPIC), as such scaling is only allowed at I picture. All subsequent P (and B) pictures must have the same picture dimensions with the preceding I picture. Therefore, all reference pictures for P or B picture can share the same surface state with the current P and B picture. Note : H/W is not processing RESPIC. Application is no longer expecting intel decoder pipeline and kernel to perform this function, it is going to be done in the video post-processing scaler or display controller scale as a separate step and controller.

All video codec surfaces must be NV12 Compliant, except JPEG. U/V vertical must be MB aligned for all video codec (further contrained for field picture), but JPEG can be block aligned. All video codec and JPEG uses Tiled - Y format only, for uncompressed pixel surfaces.

Even for JPEG planar 420 surface, application may provide only 1 buffers, but there is still only one single surface state for all of them. If IMC equal to 1, 2, 3 or 4, U and V have the pitch same as Y. And U and V will have different offset, each offset is block aligned.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_COMMON Format: OpCode
	26:24	**Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpA** Default Value: 0h Format: OpCode
	20:16	**SubOpB** Default Value: 1h Format: OpCode

MFX_SURFACE_STATE

15:12 **Reserved**

Access:	RO
Format:	MBZ

11:0 **DWord Length**

Format:	=n

Value	Name	Description
4h	DWORD_COUNT_n **[Default]**	Excludes DWord (0,1)

1

31:4 **Reserved**

Access:	RO
Format:	MBZ

3:0 **Surface Id**

Format:	U4

Value	Name	Description
0100b	Source Input Picture (encoder)	8-bit uncompressed data
0101b	Reconstructed Scaled Reference Picture	8-bit data

2

31:18 **Height**

Format:	U14-1

This field specifies the height of the Picture in units of pixels/residuals. For PLANAR surface formats, this field indicates the height of the Y (luma) plane. Note : Video Codecs must program less than and equal to 4K.(In future, it will be ideal to have this field define in a WORD boundary.)AVC - multiple of 2 MB rows for field pictureVC1 - multiple of 4 pixels for field pictureMPEG2 - multiple of 2 MB rows for field pic JPEG - multiple of integral MCU (8 or 16 pixels) per picture

Value	Name	Description
[0,16383]		representing heights [1,16384]

Programming Notes

- For AVC : For frame picture is a multiple of 16; for field picture is a multiple of 32
- For VC1 : For progressive frames, the frame height and frame width is a multiple of 2 pixels. For interlaced frames, the frame height shall be a multiple of 4 pixels, and its width is a multiple of 2 pixels, based on a PLANAR_420 surface.

17:4 **Width**

Format:	U14-1

This field specifies the width of the Picture in units of pixels/residuals. For PLANAR surface formats, this field indicates the width of the Y (luma) plane.

MFX_SURFACE_STATE

Value	Name	Description
[0,16383]		representing widths [1,16384]

Programming Notes
- The Width specified by this field multiplied by the pixel size in bytes must be less than or equal to the surface pitch (specified in bytes via the Surface Pitch field). - Width (field value + 1) must be a multiple of 2 for PLANAR_420, - MFX HW does not use this field, the picture width is read from IMG State instead, because this field may not equal to the actual picture width. This field is used by the KMD to allocate surface in GTT.

3:2 **Reserved**

Access:	RO
Format:	MBZ

1:0 **Cr(V)/Cb(U) Pixel Offset V Direction**

Format:	U0.2

Specifies the distance to the U/V values with respect to the even numbered Y channels in the V direction

Programming Notes
This field is ignored for all formats except PLANAR_420_8

3 31:28 **Surface Format**

Format:	U4

Specifies the format of the surface. All of the Y and G channels will use table 0 and all of the Cr/Cb/R/B channels will use table 1. Usage: For 420 planar YUV surface, use 4; for monochrome surfaces, use 12. For monochrome surfaces, hardware ignores control fields for Chroma planes. This field must be set to 4 - PLANAR_420_8, or 12 - Y8_UNORMNot used for MFX, and is ignored. But for JPEG decoding, this field should be programmed to the same format as JPEG_PIC_STATE. For video codec, it should set to 4 always.

Value	Name	Description
0	YCRCB_NORMAL	
1	YCRCB_SWAPUVY	
2	YCRCB_SWAPUV	
3	YCRCB_SWAPY	
4	PLANAR_420_8	(NV12, IMC1,2,3,4, YV12)
5	PLANAR_411_8	Deinterlace Only
6	PLANAR_422_8	Deinterlace Only
7	STMM_DN_STATISTICS	Deinterlace Only
8	R10G10B10A2_UNORM	Sample_8x8 Only
9	R8G8B8A8_UNORM	Sample_8x8 Only

MFX_SURFACE_STATE

10	R8B8_UNORM (CrCb	Sample_8x8 Only
11	R8_UNORM (Cr/Cb)	Sample_8x8 Only
12	Y8_UNORM	Sample_8x8 Only

27 **Interleave Chroma**

Format:	Enable

This field indicates that the chroma fields are interleaved in a single plane rather than stored as two separate planes. This field is only used for PLANAR surface formats. For AVC/VC1/MPEG VLD and IT modes : set to Enable to support interleave U/V only. For JPEG : set to Disable for all formats (including 4:2:0) - because JPEG does not support NV12. (This field is needed only if JPEG will support NV12; otherwise is ignored.)

Value	Name
1	Enable
0	Disable

26:22 **Compression Format**

Format:	Media Compression Format
Format:	Render Compression Format

Specifies the compression format.

21:20 **Reserved21_20**

19:3 **Surface Pitch**

Format:	U17-1

This field specifies the surface pitch in (#Bytes).

Value	Name
[0,131071]	

Programming Notes
For tiled surfaces, the pitch must be a multiple of the tile width (i.e.128 bytes aligned). If Half Pitch for Chroma is set, this field must be a multiple of two tile widths for tiled surfaces, or a multiple of 2 bytes for linear surfaces. For Y-tiled surfaces: Range = [127, 131071] to [128B,128KB] = [1 tile, 1024 tiles]
For TileYF and TileYS surfaces, the range is dependent on the Cu parameter (refer to Memory Data Formats section for the definition of the Cu parameter depending on the case). The range in bytes is [2$^{\text{Cu}}$-1, 131071] -> [(2$^{\text{Cu}}$)B, 128KB] = [1 tile, 128KB/(2$^{\text{Cu) tiles}}$] The field specifies the surface pitch in (#Bytes - 1)

MFX_SURFACE_STATE

If Media Memory Compression is enabled, the following max pitch size restriction must be honored. For larger resolution, Media Memory compression Must be disabled.

Tiling Mode	Pixel Format	Max Frame Width (bytes)	Max Frame Width (pixels)	Max Pitch (bytes)
Legacy 4K	8bpp	16k	16k	16k + 127
	16bpp	16k	8k	16k + 127
	32bpp	16k	4k	16k + 127
	64bpp	16k	2k	16k + 127
	128bpp	16k	1k	16k + 127
TileYF	8bpp	8k	8k	8k + 63
	16bpp	16k	8k	16k + 127
	32bpp	16k	4k	16k + 127
	64bpp	16k	2k	16k + 255
	128bpp	16k	1k	16k + 255
TileYS	8bpp	16k	16k	16k + 255
	16bpp	16k	8k	16k + 511
	32bpp	16k	4k	16k + 511
	64bpp	16k	2k	16k + 1023
	128bpp	16k	1k	16k + 1023

2 **Half Pitch for Chroma**

Format:	Enable

(This field must be set to Disable) This field indicates that the chroma plane(s) will use a pitch equal to half the value specified in the Surface Pitch field. This field is only used for PLANAR surface formats. This field is ignored by MFX (unless we support YV12)

1:0 **TileMode**

Value	Name
0	Linear
1	TileYS(64K)
2	TileX
3	TileF

4 31 **Reserved**

Access:	RO
Format:	MBZ

MFX_SURFACE_STATE

	30:16	**X Offset for U(Cb)** Format: U15 This field specifies the horizontal offset in pixels from the Surface Base Address to the start (origin) of the U(Cb) plane or the interleaved UV plane if Interleave Chroma is enabled. This field is only used for PLANAR surface formats. This field must be set to zero. X Offset for U(Cb) in pixel (This field must be zero for NV12 and IMC 1 and 3) **Programming Notes** For PLANAR_420 and PLANAR_422 surface formats, this field must be zero.
	15	**Reserved** Access: RO Format: MBZ
	14:0	**Y Offset for U(Cb)** Format: U15 This field specifies the vertical offset in rows from the Surface Base Address to the start (origin) of the U(Cb) plane or the interleaved UV plane if Interleave Chroma is enabled. This field is only used for PLANAR surface formats. **Programming Notes** For PLANAR_420 and PLANAR_422 surface formats, this field must be multiple of 16 pixels - i.e. multiple MBs. For JPEG, this field must be a multiple of 16 pixels.
5	31:29	**Reserved** Access: RO Format: MBZ
	28:16	**X Offset for V(Cr)** Format: U13 This field must be zero for NV12 and IMC 1 and 3 This field specifies the horizontal offset in pixels from the Surface Base Address to the start (origin) of the V(Cr) plane. This field is only used for PLANAR surface formats with Interleave Chroma disabled. **Programming Notes** For PLANAR_420 and PLANAR_422 surface formats, this field must indicate an even number of pixels.
	15:0	**Y Offset for V(Cr)** Format: U16 This field specifies the vertical offset in rows from the Surface Base Address to the start (origin) of the V(Cr) plane. This field is only used for PLANAR surface formats with Interleave Chroma disabled. This field is ignored by all video codec, only used by JPEG.

MFX_SURFACE_STATE		
		Programming Notes For PLANAR_420 surface formats, this field must be multiple of 16 pixels - i.e. multiple MBs. For JPEG, this field must be a multiple of 16 pixels.

MFX_VC1_DIRECTMODE_STATE

<table>
<tr><th colspan="3">MFX_VC1_DIRECTMODE_STATE</th></tr>
<tr><td colspan="3">Source: VideoCS
Length Bias: 2
Exists If: //VC1 decoding in VLD modes</td></tr>
<tr><td colspan="3">This is a picture level command and should be issued only once, even for a multi-slices picture. There is only one DMV buffer for read (when processing a B-picture) and one for write (when processing a P-Picture). Each DMV record is 64 bits per MB, to store the top and bottom field MVs (32-bit MVx,y each).</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
Default Value: 3h PARALLEL_VIDEO_PIPE
Format: OpCode</td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline</b>
Default Value: 2h MFX_VC1_DIRECTMODE_STATE
Format: OpCode</td></tr>
<tr><td></td><td>26:24</td><td><b>Media Command Opcode</b>
Default Value: 2h VC1_COMMON
Format: OpCode</td></tr>
<tr><td></td><td>23:21</td><td><b>SubOpcode A</b>
Default Value: 0h
Format: OpCode</td></tr>
<tr><td></td><td>20:16</td><td><b>SubOpcode B</b>
Default Value: 2h
Format: OpCode</td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b>
Access: RO
Format: MBZ</td></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b>
Default Value: 0005h Excludes DWord (0,1)
Format: =n</td></tr>
<tr><td>1..2</td><td>63:0</td><td><b>Direct MV Write Buffer - Base Address</b>
Format: SplitBaseAddress64ByteAligned
This field provides the base address of the DMV write buffer to store the motion vectors decoded in the current picture. It is a private buffer used by the MPR hardware only. Its content is not accessed by software. This buffer must be 64-byte cacheline aligned. The write buffer size is 557,056 bytes for 1 frame. Scalable with frame height, but do not scale with frame width as the hardware assumes frame width (in MBs) fixed at 128 (smallest power of 2 value larger than 120 - 1920x1088 screen resolution). This field is only valid for a P picture</td></tr>
</table>

MFX_VC1_DIRECTMODE_STATE		
3	31:0	**Direct MV Write Buffer - Attributes** Format: **MemoryAddressAttributes**
4..5	63:0	**Direct MV Reference Buffer - Base Address** Format: **SplitBaseAddress64ByteAligned** This field provides the base address of the DMV buffer for reference picture. It is a private buffer used by the MPR hardware only. Its content is not accessed by software. All these buffers must be 64-byte cacheline aligned. This field is only valid for a B picture.
6	31:0	**Direct MV Reference Buffer - Attributes** Format: **MemoryAddressAttributes**

MFX_VC1_PRED_PIPE_STATE

MFX_VC1_PRED_PIPE_STATE		
Source:	VideoCS	
Length Bias:	2	
This command is used to set the operating states of the MFD Engine beyond the BSD unit. It is used with both VC1 Long and Short format. Driver is responsible to take the intensity compensation enable signal, the LumScale and the LumShift provided from the VC1 interface, and maintain a history of these values for reference pictures. Together with these three parameters specified for the current picture being decoded, driver will derive and supply the above sets of LumScaleX, LumShiftX and intensity compensation enable (single or double, forward or backward) signals. H/W is responsible to take these state values, and use them to build the lookup table (including the derivation of iScale and iShift) for remapping the reference frame pixels, as well as performing the actual pixel remapping calculations/process.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_VC1_PRED_PIPE_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 1h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 0004h Excludes DWord (0,1) Format: =n
1	31:16	**Reserved** Access: RO Format: MBZ

MFX_VC1_PRED_PIPE_STATE

	15:14	**vin_intensitycomp_Double_FWDen** Format: U2 for forward reference picture only, to enable top or/and bottom of the reference field enable for single compensation. For frame, may only need one bit. This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	13:12	**vin_intensitycomp_Double_BWDen** Format: U2 for backward reference picture only, no double for backward reference. This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	11:10	**vin_intensitycomp_Single_FWDen** Format: U2 for forward reference picture only, to enable top or/and bottom of the reference field enable for single compensation. For frame, may only need one bit. This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	9:8	**vin_intensitycomp_Single_BWDen** Format: U2 for backward reference picture only, no double for backward reference. This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:4	**Reference Frame Boundary Replication Mode** Format: U4 This is a bit field with each bit indicating the corresponding picture's boundary replication mode. Bit 11: reference 3Bit 10: reference 2Bit 9: reference 1Bit 8: reference 00 = progressive frame replication1 = interlace frame replication This field is maintained and provided by driver for both long and short VC1 interface format.
	3:0	**Reserved** Access: RO Format: MBZ
2	31:30	**Reserved** Access: RO Format: MBZ

MFX_VC1_PRED_PIPE_STATE

	29:24	**LumShift2- single - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	23:22	**Reserved** Access: RO Format: MBZ
	21:16	**LumShift1 - single - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	15:14	**Reserved** Access: RO Format: MBZ
	13:8	**LumScale2 - single - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:6	**Reserved** Access: RO Format: MBZ
	5:0	**LumScale1 - Single - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
3	31:30	**Reserved** Access: RO Format: MBZ

MFX_VC1_PRED_PIPE_STATE

	29:24	**LumShift2- double - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	23:22	**Reserved** Access: RO Format: MBZ
	21:16	**LumShift1 - double -FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	15:14	**Reserved** Access: RO Format: MBZ
	13:8	**LumScale2 - double - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:6	**Reserved** Access: RO Format: MBZ
	5:0	**LumScale1 - double - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
4	31:30	**Reserved** Access: RO Format: MBZ

MFX_VC1_PRED_PIPE_STATE

	29:24	**LumShift2- single - BWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	23:22	**Reserved** Access: RO Format: MBZ
	21:16	**LumShift1 - single - BWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	15:14	**Reserved** Access: RO Format: MBZ
	13:8	**LumScale2 - single - BWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:6	**Reserved** Access: RO Format: MBZ
	5:0	**LumScale1 - Single - BWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
5	31:30	**Reserved** Access: RO Format: MBZ

MFX_VC1_PRED_PIPE_STATE

	29:24	**LumShift2- double - BWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	23:22	**Reserved** Access: RO Format: MBZ
	21:16	**LumShift1 - double -BWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	15:14	**Reserved** Access: RO Format: MBZ
	13:8	**LumScale2 - double - BWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:6	**Reserved** Access: RO Format: MBZ
	5:0	**LumScale1 - double - BWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

Source:	VideoCS
Length Bias:	2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Video Codec Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VP8 Format: OpCode
	23:21	**Sub Opcode A** Default Value: 2h VP8 Common Format: OpCode
	20:16	**Sub Opcode B** Default Value: 3h MFX_VP8_BSP_BUF_BASE_ADDR_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Format: =n **Value** | **Name** | **Description** 000h | Excludes DWord (0,1) **[Default]** | A special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware." 008h | | Used for normal encode mode
1..2	63:0	**Frame Header - Base Address** Format: SplitBaseAddress64ByteAligned 64 byte aligned, 48-bit Abs. Address StreamIn Surface **Note:**The format is linear vs. tile for better performance.

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

3	31:0	**Frame Header - Attributes** Format: MemoryAddressAttributes
4..5	63:0	**Intermediate Buffer - Base Address** Format: SplitBaseAddress64ByteAligned 64 byte aligned, 48-bit AbsAddr StreamIn Surface **Note:** The format is linear vs. tile for better performance.
6	31:0	**Intermediate Buffer - Attributes** Format: MemoryAddressAttributes
7..14	255:0	**Intermediate Buffer Partition Offset** **Programming Notes** All **Intermediate Buffer Partition-[i] Offset** (i = 1 to 8) and **Intermediate Buffer Max Size** need to be cacheline aligned (64Byte aligned).
15	31:0	**Intermediate Buffer Max Size** Format: U32
16..17	63:0	**Final Frame - Base Address** Format: SplitBaseAddress64ByteAligned 64 byte aligned, 48-bit AbsAddr StreamIn Surface **Note:** The format is linear vs. tile for better performance.
18	31:0	**Final Frame - Attributes** Format: MemoryAddressAttributes
19	31:6	**Reserved** Access: RO Format: MBZ
	5:0	**Final Frame Byte Offset** Format: U6 Specify byte offset within a 64-byte cacheline where the bitstream should be inserted at.
20..21	63:0	**Streamout - Base Address** Format: SplitBaseAddress64ByteAligned 64 byte aligned,48-bit AbsAddr StreamIn Surface **Note:** The format is linear vs. tile for better performance.
22	31:0	**Streamout - Attributes** Format: MemoryAddressAttributes
23..24	63:0	**Coeff Probs StreamIn Surface - Base Address** Format: SplitBaseAddress64ByteAligned

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

		64 byte aligned, 48-bit AbsAddr StreamIn Surface **Note:** The format is linear vs. tile for better performance.
25	31:0	**Coeff Probs StreamIn Surface - Attributes** Format: **MemoryAddressAttributes**
26..27	63:0	**Token Statistics Surface - Base Address** Format: **SplitBaseAddress64ByteAligned** 64 byte aligned, 48-bit Abs. Address StreamIn Surface **Note:**The format is linear vs. tile for better performance.
28	31:0	**Token Statistics Surface - Attributes** Format: **MemoryAddressAttributes**
29..30	63:0	**MPC RowStore Surface - Base Address** Format: **SplitBaseAddress64ByteAligned** Abs. Address StreamIn/StreamOut Surface. **Note:** The format is linear vs. tile for better performance. GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and MBZ.
31	31:0	**MPC RowStore Surface - Attributes** Format: **MemoryAddressAttributes**

MFX_VP8_Encoder_CFG

MFX_VP8_Encoder_CFG		
Source:	VideoCS	
Length Bias:	2	
This must be the very first command to issue after the surface state, the pipe select and base address setting commands and must be issued before MFX_VP8_PIC_STATE.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Video Codec Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VP8 Format: OpCode
	23:21	**Sub Opcode A** Default Value: 2h VP8 Common Format: OpCode
	20:16	**Sub Opcode B** Default Value: 1h MFX_VP8_ENCODER_CFG Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Format: =n **Value** | **Name** | **Description** 000h | Excludes DWord (0,1) **[Default]** | A special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware." 01Dh | | Used for normal encode mode
1	31:11	**Reserved** Access: RO Format: MBZ

MFX_VP8_Encoder_CFG

	10	**VBSPunitPowerClock Gating Disable** Format: U1 VBSPunit Power Clock Gating Disable.
	9	**Compressed Bitstream Output Disable** Format: U1 Disable Compressed Bitstream Output.**(Both Final Bitstream and Intermediate bit buffer)**
	8	**Finer BRC Enable** Format: U1 Enable Finer BRC Feature.
	7	**Per Segment Delta Qindex / LoopFilter Disable** Format: U1 Disable Per Segment Delta Qindex / Loop Filter in Rate Control.
	6	**Rate Control Initial Pass** Format: U1 **Value** — **Name** 1 — Initial pass 0 — Subsequence Pass(es)
	5	**Skip Final Bitstream when Over / Under flow** Format: U1 Skip Final Bitstream conditionally on Over/Under flow in rate control and intermediate Bit Buffer Overrun.
	4	**Update Segment Feature Data Flag** Exists If: //VP8 Encoder Format: U1 Enable for Frame Header per Segment Quantizer / LoopFilter Update
	3	**Bitstream Statistics Output Enable** Enable Bitstream Statistics Output at Memory Surface in MFX_VBSP_BUF_ADDR_STATE DW[26:28]
	2	**Token Statistics Output Enable** Enable Token Statistics Output at Memory Surface in MFX_VBSP_BUF_ADDR_STATE DW[26:28]
	1	**Final Bitstream Output Disable** Format: U1 Disable Final Bitstream Output.
	0	**Performance Counter Enable** Format: U1 Enable Performance Counter in Streamout.

<table>
<tr><th colspan="3">MFX_VP8_Encoder_CFG</th></tr>
<tr><td>2</td><td>31:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7</td><td><b>Qindex_Clamp_High_mask for overflow</b><table><tr><td>Format:</td><td>U1</td></tr></table>If current frame is overflow and this mask is set, it would mask out MFX_VP8_Img_Status register. DW1.bit1. In another word, subsequent passes would be skipped.</td></tr>
<tr><td></td><td>6</td><td><b>Qindex_Clamp_High_mask for underflow</b><table><tr><td>Format:</td><td>U1</td></tr></table>If current frame is underflow and this mask is set, it would mask out MFX_VP8_Img_Status register. DW1.bit0. In another word, subsequent passes would be skipped</td></tr>
<tr><td></td><td>5</td><td><b>Final Bistream Buffer Overrun Enable Mask</b><table><tr><td>Format:</td><td>U1</td></tr></table>Enable Final Bitstream Buffer Overrun detection feature.</td></tr>
<tr><td></td><td>4</td><td><b>Intermediate Bit Buffer Overrun Enable Mask</b><table><tr><td>Format:</td><td>U1</td></tr></table>Enable Intermediate Bit Buffer Overrun detection feature.</td></tr>
<tr><td></td><td>3</td><td><b>Max Intra MB Bit Count Check Enable Mask</b><table><tr><td>Format:</td><td>U1</td></tr></table>Enable Max. Intra MB bit count check in Streamout.</td></tr>
<tr><td></td><td>2</td><td><b>Max Inter MB Bit Count Check Enable Mask</b><table><tr><td>Format:</td><td>U1</td></tr></table>Enable Max. Inter MB bit count check in Streamout.</td></tr>
<tr><td></td><td>1</td><td><b>Min Frame Bit Count Rate Control Enable Mask</b><table><tr><td>Format:</td><td>U1</td></tr></table>Enable Min. Frame Rate Control. This is a mask bit controlling if the condition of frame level bit count is less than or equal to FrameBitRateMin.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>1</td><td></td><td>If (Total Frame Level Bit Counter) =< (Frame Bit Rate Minimum limit)Set bit[0] and bit[1] of MFX_VP8_IMAGE_STATUS Control Register.</td></tr><tr><td>0</td><td></td><td>Do not update bit[0] of MFX_VP8_IMAGE_STATUS Control Register.</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>Max Frame bit count Rate Control Enable Mask</b><table><tr><td>Format:</td><td>U1</td></tr></table>Enable Max. Frame Rate Control. This is a mask bit controlling if the condition of frame level bit count is greater than or equal to FrameBitRateMax.</td></tr>
</table>

<table>
<tr><th colspan="3">MFX_VP8_Encoder_CFG</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td></td><td>If (Total Frame Level Bit Counter) >= (Frame Bit Rate Maximum Limit)Set bit[0] and bit[1] of MFX_VP8_IMAGE_STATUS control register.</td></tr>
<tr><td>0</td><td></td><td>Do not update bit[0] of MFX_VP8_IMAGE_STATUS control register.</td></tr>
</table>
</td></tr>
<tr><td>3</td><td>31:28</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>27:16</td><td><b>Max Intra MB Bit Count Limit</b>
<table>
<tr><td>Format:</td><td>U12</td></tr>
</table>
12-bit bit count for Max Intra MB Limit.</td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>11:0</td><td><b>Max Inter MB bit count</b>
<table>
<tr><td>Format:</td><td>U12</td></tr>
</table>
12-bit bit count for Max Inter MB Limit.</td></tr>
<tr><td>4</td><td>31</td><td><b>Frame Bitrate Min Unit Mode</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
This field is the Frame Bitrate Minimum Limit Units.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Compatibility Mode</td><td>Frame BitRate Min Unit is in old mode <b>(128b/16Kb)</b></td></tr>
<tr><td>1h</td><td>New Mode</td><td>Frame BitRate Min Unit is in new mode <b>(32byte/4Kb)</b></td></tr>
</table>
</td></tr>
<tr><td></td><td>30</td><td><b>Frame Bit Rate Min Unit</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
<i>This field is Frame Bitrate Minimum Mode.</i>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>32-B</td></tr>
<tr><td>1</td><td>4-KB</td></tr>
</table>
</td></tr>
<tr><td></td><td>29:16</td><td><b>Frame Bit Rate Min</b>
<table>
<tr><td>Format:</td><td>U14</td></tr>
</table>
If either BRC Underflow or overflow is enabled. Frame Bit Rate Min and Frame Bit Rate Max need to be programmed with unambiguous values</td></tr>
<tr><td></td><td>15</td><td><b>Frame Bitrate Max Unit Mode</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
This field is the Frame Bitrate Maximum Limit Units.</td></tr>
</table>

<table>
<tr><th colspan="3">MFX_VP8_Encoder_CFG</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Compatibility Mode</td><td>Frame BitRate Max Unit is in old mode <b>(128b/16Kb)</b></td></tr>
<tr><td>1h</td><td>New Mode</td><td>Frame BitRate Max Unit is in new mode <b>(32byte/4Kb)</b></td></tr>
</table>
</td></tr>
<tr><td></td><td>14</td><td><b>Frame Bit Rate Max Unit</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
<i>This field is Frame Bitrate Maximum Mode</i>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>32-B</td></tr>
<tr><td>1</td><td>4-KB</td></tr>
</table>
</td></tr>
<tr><td></td><td>13:0</td><td><b>Frame Bit Rate Max</b>
<table>
<tr><td>Format:</td><td>U14</td></tr>
</table>
If either BRC Underflow or overflow is enabled. Frame Bit Rate Min and Frame Bit Rate Max need to be programmed with unambiguous values
</td></tr>
<tr><td>5</td><td>31:24</td><td><b>Frame Delta QIndex Max[3]</b>
This field is the Frame level delta Qindex for total bit-count above FrameBitRateMax - First 1/8 Region.
This field is used to calculate the suggested Frame Qindex into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame exceeds FrameBitRateMax but is within 1/8 of FrameBitRateMaxDelta above <b>FrameBitRateMax</b>;i.e., In the range of (FrameBitRateMax, (FrameBitRateMax + FrameBitRateMaxDelta » 3)].</td></tr>
<tr><td></td><td>23:16</td><td><b>Frame DeltaQ Index Max[2]</b>
This field is the Frame level delta Qindex for bit-count above FrameBitRateMax - Above 1/8 and Below 1/4.
This field is used to calculate the suggested Frame Qindex into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between 1/8 and 1/4 of FrameBitRateMaxDelta above <b>FrameBitRateMax</b>;i.e., In the range of ((FrameBitRateMax + FrameBitRateMaxDelta » 3), (FrameBitRateMax+ FrameBitRateMaxDelta » 2)].</td></tr>
<tr><td></td><td>15:8</td><td><b>Frame Delta QIndex Max[1]</b>
This field is the Frame level delta QINDEX for bit-count above FrameBitRateMax - Above 1/4 and Below 1/2.
This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between 1/4 and 1/2 of FrameBitRateMaxDelta above <b>FrameBitRateMax</b>;i.e., In the range of [(FrameBitRateMax+ FrameBitRateMaxDelta » 2), (FrameBitRateMax+ FrameBitRateMaxDelta » 1)].</td></tr>
<tr><td></td><td>7:0</td><td><b>Frame Delta QIndex Max [0]</b>
This field is the Frame level delta QINDEX for bit-count above FrameBitRateMax - Above 1/2.</td></tr>
</table>

MFX_VP8_Encoder_CFG

		This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is above FrameBitRateMax by more than half the distance of **FrameBitRateMax**;i.e., In the range of [(FrameBitRateMax + FrameBitRateMaxDelta » 1), (Infinite)].
6	31:24	**Frame Delta QIndex Min[3]** This field is the Frame level delta QINDEX for total bit-count below FrameBitRateMin - First 1/8 Region. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is less than FrameBitRateMin and greater than or equal to 1/8 the distance of FrameBitRateMinDelta from **FrameBitRateMin**; i.e., In the range of [(FrameBitRateMin - FrameBitRateMinDelta » 3), FrameBitRateMin].
	23:16	**Frame Delta QIndex Min[2]** This field is the Frame level delta QINDEX for bit-count below FrameBitRateMin - Below 1/ 8 and Above 1/4. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between one-eighth and quarter the distance of FrameBitRateMinDelta from **FrameBitRateMin**;i.e., In the range of [(FrameBitRateMin - FrameBitRateMinDelta » 2), (FrameBitRateMin- FrameBitRateMinDelta » 3)].
	15:8	**Frame Delta QIndex Min[1]** This field is the Frame level delta QINDEX for bit-count below FrameBitRateMin- Below 1/4 and Above 1/2. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between quarter and half the distance of FrameBitRateMinDelta from **FrameBitRateMin**;i.e., In the range of [(FrameBitRateMin - FrameBitRateMinDelta » 1), (FrameBitRateMin - FrameBitRateMinDelta » 2)].
	7:0	**Frame Delta QIndex Min[0]** This field is the Frame Level Delta QINDEX for bit-count below FrameBitRateMin - Below 1/2. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is below FrameBitRateMin by more than half the distance of **FrameBitRateMin**;i.e., In the range of [0, (FrameBitRateMin - FrameBitRateMinDelta » 1).
7	31:0	**Per Segment Frame Delta QIndex Max[1]**
8	31:0	**Per Segment Frame Delta QIndex Min[1]**
9	31:0	**Per Segment Frame Delta QIndex Max[2]**
10	31:0	**Per Segment Frame Delta QIndex Min[2]**
11	31:0	**Per Segment Frame Delta QIndex Max[3]**
12	31:0	**Per Segment Frame Delta QIndex Min[3]**

MFX_VP8_Encoder_CFG

13	31:24	**Frame Delta Loop Filter Max[3]** Format: U8 This field is the Frame level delta LoopFilter for total bit-count above FrameBitRateMax - First 1/8 region. This field is used to calculate the suggested Frame LoopFilter into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame exceeds FrameBitRateMax but is within 1/8 of FrameBitRateMaxDelta above FrameBitRateMax.i.e., in the range of (FrameBitRateMax, (FrameBitRateMax+ FrameBitRateMaxDelta » 3)].
	23:16	**Frame Delta Loop Filter Max[2]** Format: U8 This field is the Frame level delta LoopFilter for bit-count above FrameBitRateMax - Above 1/8 and Below 1/4. This field is used to calculate the suggested Frame LoopFilter into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between 1/8 and 1/4 of FrameBitRateMaxDelta above FrameBitRateMax.i.e., in the range of ((FrameBitRateMax + FrameBitRateMaxDelta » 3) and (FrameBitRateMax + FrameBitRateMaxDelta » 2)].
	15:8	**Fram eDelta Loop Filter Max[1]** Format: U8 This field is the Frame level delta LOOPFILTER for bit-count above FrameBitRateMax - Above1/ 4 and Below 1/2. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between 1/4 and1/2 of FrameBitRateMaxDelta above FrameBitRateMax.i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta » 2) and (FrameBitRateMax+ FrameBitRateMaxDelta » 1)].
	7:0	**Frame Delta Loop Filter Max[0]** Format: U8 This field is the Frame level delta LOOPFILTER for bit-count above FrameBitRateMax - Above 1/2. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is above FrameBitRateMax by more than half the distance of FrameBitRateMaxDelta.i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta » 1), infinite).
14	31:24	**Frame Delta Loop Filter Min[3]** Format: U8 This field is the Frame level delta LOOPFILTER for total bit-count below FrameBitRateMin - First 1/8 region. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is less than FrameBitRateMin and greater than or equal to 1/8 the distance of FrameBitRateMinDelta from FrameBitRateMin.i.e., in the range of

MFX_VP8_Encoder_CFG

		[(FrameBitRateMin - FrameBitRateMinDelta » 3), FrameBitRateMin).
	23:16	**Frame Delta Loop Filter Min[2]** Format: U8 This field is the Frame level delta LOOPFILTER for bit-count below FrameBitRateMin - Below 1/ 8 and Above 1/4. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between one-eighth and quarter the distance of FrameBitRateMinDelta from FrameBitRateMin.i.e., in the range of [(FrameBitRateMin - FrameBitRateMinDelta » 2), (FrameBitRateMin - FrameBitRateMinDelta » 3)).
	15:8	**Frame Delta Loop Filter Min[1]** Format: U8 This field is the Frame level delta LOOPFILTER for bit-count below FrameBitRateMin- Below 1/4 and Above 1/2. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between quarter and half the distance of FrameBitRateMinDelta from FrameBitRateMin.i.e., in the range of [(FrameBitRateMin - FrameBitRateMinDelta » 1) and (FrameBitRateMin - FrameBitRateMinDelta » 2)).
	7:0	**Frame Delta Loop Filter Min[0]** Format: U8 This field is the Frame Level Delta LOOPFILTER for bit-count below FrameBitRateMin - Below 1/ 2. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is below FrameBitRateMin by more than half the distance of FrameBitRateMinDelta.i.e., in the range of [0, (FrameBitRateMin - FrameBitRateMinDelta » 1).
15	31:0	**Per Segment Frame Delta LoopFilter Max[1]**
16	31:0	**Per Segment Frame Delta LoopFilter Min[1]**
17	31:0	**Per Segment Frame Delta LoopFilter Max[2]**
18	31:0	**Per Segment Frame Delta LoopFilter Min[2]**
19	31:0	**Per Segment Frame Delta LoopFilter Max[3]**
20	31:0	**Per Segment Frame Delta LoopFilter Min[3]**
21	31	**Reserved** Access: RO Format: MBZ
	30:16	**FrameBitRateMinDelta** Format: U15 This field is used to select the frame delta QINDEX when FrameBitRateMin Is exceeded. It shares the same FrameBitrateMinUnit.

<table>
<tr><th colspan="3">MFX_VP8_Encoder_CFG</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>[0-4095]</td><td></td><td>When FrameBitrateMinUnit is in Bytes, this range is in Bytes.
When FrameBitrateMinUnit is in KB, this range is in KB units.</td></tr></table></td></tr>
<tr><td></td><td>15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:0</td><td><b>Frame Bit Rate Max Delta</b><table><tr><td>Format:</td><td>U15</td></tr></table>This field is used to select the frame delta QINDEX when FrameBitRateMax Is exceeded. It shares the same FrameBitrateMaxUnit.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>[0-4095]</td><td></td><td>When FrameBitrateMinUnit is in Bytes, this range is in Bytes.
When FrameBitrateMinUnit is in KB, this range is in KB units.</td></tr></table></td></tr>
<tr><td>22</td><td>31:24</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>23</td><td><b>Show Frame</b><table><tr><td>Format:</td><td>U1</td></tr></table>VP8 Frame Tag, Show Frame Field</td></tr>
<tr><td></td><td>22:20</td><td><b>Bitstream Format Version</b><table><tr><td>Format:</td><td>U3</td></tr></table>VP8 Frame Tag, Version Field</td></tr>
<tr><td></td><td>19:18</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>17:16</td><td><b>Min Frame WSize Unit</b><table><tr><td>Format:</td><td>U2</td></tr></table><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Compatibility Mode</td><td>MinFrameWSizeUnit is in old mode (128b/16Kb)</td></tr><tr><td>1h</td><td>New Mode</td><td>MinFrameWSizeUnit is in new mode (32byte/4Kb)</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Min Frame WSize</b><table><tr><td>Exists If:</td><td>//Encoder Only</td></tr></table><table><tr><td>This field (in Word, 16-bit) is specified to compensate for Intel Rate Control.</td></tr><tr><td>Zero padding would be performed.</td></tr></table></td></tr>
</table>

MFX_VP8_Encoder_CFG

23	31:16	**Vertical_Size_Code** Format: U16 Frame Tag Vertical Size Code, composed of{VerticalScale[15:14], FrameHeight[13:0]}
	15:0	**Horizontal_Size_Code** Format: U16 Frame Tag Horizontal Size Code, composed of{HorizontalScale[15:14], FrameWidth[13:0]}
24	31:0	**Frame Header Bit Count** Format: U32 Binarized Header Bit Count.
25	31:0	**Frame Header Bin Buffer Qindex Update Pointer** Format: U32 Binarized Header Qindex Update PointerIf Segment Enabled and UpdateSegmentFeature enabled, 4 per segment Qindices would be updated in Binarized header (Only ABS mode supported).Else Base Qindex would be updated
26	31:0	**Frame Header Bin Buffer LoopFilter Update Pointer** Format: U32 Binarized Header LoopFilter Update PointerIf Segment Enabled and UpdateSegmentFeature enabled, 4 per segment LoopFilters would be updated in Binarized header (Only ABS mode supported).ElseBase LoopFilter would be updated.
27	31:0	**Frame Header Bin Buffer Token Update Pointer** Format: U32 Binarized Header TokenUpdate Pointer
28	31:0	**Frame Header Bin Buffer MVUpdate Pointer** Format: U32 Binarized Header MVUpdate Pointer.
29 **Programming Notes:** The only value permitted for CV7 through CV0 is 0xf	31:28	**ClampValues - CV7**
	27:24	**CV6**
	23:20	**CV5**
	19:16	**CV4**
	15:12	**CV3**
	11:8	**CV2**
	7:4	**CV1**
	3:0	**CV0 - Clamp Value 0** Format: U4 If the magnitude of coefficients at locations assigned with CV0 (mapping shown below) exceeds 2CV0-1, they are replaced with 2CV0-1. For coefficients at

MFX_VP8_Encoder_CFG

locations marked as 'none', no clamping is performed. The following mappings are only applied to luma and chroma blocks\subblocks containing AC coefficiencts (blocks\sublocks with only DC coeffs will not be clamped).

For 4x4 frame block, each coefficient is mapped to one of the eight CV values as following:

none	CV7	CV5	CV4
CV7	CV6	CV4	CV3
CV5	CV4	CV2	CV1
CV4	CV3	CV1	CV0

For 4x4 field block, each coefficient is mapped to one of the eight CV values as following:

none	CV6	CV3	CV1
CV7	CV6	CV3	CV1
CV5	CV4	CV2	CV0
CV5	CV4	CV2	CV0

Value	Name
0-15	

MFX_VP8_PAK_OBJECT

MFX_VP8_PAK_OBJECT

Source:	VideoCS
Length Bias:	2

The MFX_VP8_PAK_OBJECT command is the second primitive command for the VP8 Encoding Pipeline. The MV Data portion of the bitstream is loaded as indirect data object. Before issuing a MFX_VP8_PAK_OBJECT command, all VP8 MFX states need to be valid; therefore the commands used to set these states need to have been issued prior to the issue of this command. MB record must be consecutive with no gaps, hence we do not need MB(x,y) in each MB command. Internal counter will keep track of the current MB address, starting from the first MB.MFX_VP8_PAK_OBJECT command follows the MbType definition like MFD. Encoding statistical data such as the total size of the output bitstream are provided through MMIO registers. Software may access these registers through MI_STORE_REGISTER_MEM command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_VP8_PAK_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 4h VP8_ENC Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 5h DWORD_COUNT_n Format: =n
1	31:30	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">MFX_VP8_PAK_OBJECT</th></tr>
<tr><td></td><td>29</td><td><b>Enable Inline MV data</b><table><tr><td>Format:</td><td>Enable</td></tr></table>This field denotes if the MV data will be sent inline following the other inline data instead of being indirect.</td></tr>
<tr><td></td><td>28:10</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>9:0</td><td><b>Indirect PAK-MV Data Length</b><table><tr><td>Format:</td><td>U10</td></tr></table>This field provides the length in bytes of the indirect data, which contains all the MVs for the current MB (in any partitioning and subpartitioning form). A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect PAK-MV Data Start Address field is ignored. This field must have the same alignment as the Indirect PAK-MV Data Start Address. This field must be DW aligned (since each MV is 4 bytes in size).Driver has to derived this field from MVsize (MVquantity in DXVA, exact size) *4 bytes per MV.</td></tr>
<tr><td>2</td><td>31:29</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>28:0</td><td><b>Indirect PAK-MV Data Start Address Offset</b>
This field specifies the memory starting address (offset) of the MV data to be fetched into PAK Subsystem for processing. This pointer is relative to the MFC Indirect PAK-MV Object Base Address. Hardware ignores this field if indirect data is not present, i.e. the Indirect PAK-MV Data Length is set to 0. It is a Dword aligned address in all AVC encoding configuration, since each MV is 4 bytes in size.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[0,512MB)</td><td></td></tr></table></td></tr>
<tr><td>3..6</td><td>127:0</td><td><b>Inline Data</b>
All the required MB level controls and parameters for encoding are captured as Inline Data Description - VP8 PAK OBJECT. It has a fixed size of 4 DWs. Its definition is described in the next section.</td></tr>
</table>

MFX_VP8_PIC_STATE

MFX_VP8_PIC_STATE		
Source:	VideoCS	
Length Bias:	2	
This must be the very first command to issue after the surface state, the pipe select and base address setting commands and must be issued before MFX_VP8_IMG_STATE.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Video Codec Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VP8 Format: OpCode
	23:21	**Sub Opcode A** Default Value: 0h VP8 Common Format: OpCode
	20:16	**Sub Opcode B** Default Value: 0h MFX_VP8_PIC_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Format: =n **Value** | **Name** | **Description** 000h | Excludes DWord (0,1) **[Default]** | A special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware." 024h | | Used for normal decode and encode mode
1	31:24	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">MFX_VP8_PIC_STATE</th></tr>
<tr><td></td><td>23:16</td><td><b>Frame Height Minus 1</b><table><tr><td>Exists If:</td><td>//Decoder / Encoder</td></tr><tr><td>Format:</td><td>U8</td></tr></table>Picture Height in integer number of MBs minus 1, so the min pic height can be program is 16 rows of pixels.</td></tr>
<tr><td></td><td>15:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>Frame Width Minus 1</b><table><tr><td>Exists If:</td><td>//Decoder / Encoder</td></tr><tr><td>Format:</td><td>U8</td></tr></table>Picture Width in integer number of MBs minus 1, so the min pic width can be program is 16 pixels.</td></tr>
<tr><td>2</td><td>31:26</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>25:24</td><td><b>Log2 Num of Partition</b><table><tr><td>Exists If:</td><td>//Decoder / Encoder</td></tr><tr><td>Format:</td><td>U2</td></tr></table><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>1 Token partition</td></tr><tr><td>1</td><td>2 Token partition</td></tr><tr><td>2</td><td>4 Token partition</td></tr><tr><td>3</td><td>8 Token partition</td></tr></table></td></tr>
<tr><td></td><td>23:19</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>18:16</td><td><b>Deblock Sharpness Level</b><table><tr><td>Exists If:</td><td>//Decoder / Encoder</td></tr><tr><td>Format:</td><td>U3</td></tr></table>Specify the sharpness level, as one of the regular deblocking strength control parameters.<table><tr><th>Programming Notes</th></tr><tr><td>Set to 0 to disable the use of sharpness control.</td></tr></table></td></tr>
<tr><td></td><td>15:14</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

MFX_VP8_PIC_STATE

13 **Alternate Ref Pic MV SignBias Flag**

Exists If:	//Decoder / Encoder

Alternate Reference Picture MV sign bias flag, specified for non-key frame only.

12 **Golden Ref Picture MV SignBias Flag**

Exists If:	//Decoder / Encoder

Golden Reference Picture MV sign bias flag, specified for non-key frame only.

11 **Mode Reference Loop Filter Delta Enabled**

Exists If:	//Decoder / Encoder

Value	Name	Description
0		Mode or Reference Loop Filter Delta Adjustment for current frame is disabled.
1		Mode or Reference Loop Filter Delta Adjustment for current frame is enabled.

10 **MB NoCoeff SkipFlag**

Exists If:	//Decoder / Encoder

Frame level control if Skip MB (with no non-zero coefficient) is allowed or not.

Value	Name	Description
0		All MBs will have its MB level signaling mb_skip_coeff forced to 0. That is, no skip of coefficient record in the bitstream (even their values are all 0s)
1		Skip MB is enabled in the per MB record.

9 **Update MBSegment Map Flag**

Exists If:	//Decoder / Encoder

Value	Name	Description
0		Disable segmentation update
1		Enable segmentation update, and to enable reading segment_id for each MB.

8 **Segment Enable Flag**

Exists If:	//Decoder / Encoder

Value	Name	Description
0		Disable Segmentation processing in the current frame
1		Enable Segmentation processing in the current frame

7 **Segmentation ID StreamIn Enable**

Exists If:	//Decoder Only

Value	Name
0	StreamIn Disabled
1	StreamIn Enabled

MFX_VP8_PIC_STATE

Programming Notes
When 0, no input needed.

6 **Segmentation ID StreamOut Enable**

Exists If:	//Decoder Only

Value	Name
0	StreamOut Disabled
1	StreamOut Enabled

Programming Notes
When 0, no output needed.

5 **sKeyFrameFlag**

Exists If:	//Decoder / Encoder

Value	Name
0	Non-Key Frame (P-Frame)
1	Key Frame (I-Frame)

4 **DBLKFilterType**

Exists If:	//Decoder / Encoder

To specify VP8 Profile of operation.

Value	Name	Description
0		Use a full feature normal deblocking filter
1		Use a simple filter for deblocking

3:2 **Reserved**

Access:	RO
Format:	MBZ

1 **Chroma Full Pixel MC Filter Mode**

Exists If:	//Decoder / Encoder

To specify VP8 Profile of operation.

Value	Name	Description
0		Chroma MC filter operates in sub-pixel mode
1		Chroma MC filter only operates in full pixel position, i.e. no sub-pixel interpolation.

0 **MC Filter Select**

Exists If:	//Decoder / Encoder

To specify VP8 Profile of operation.

MFX_VP8_PIC_STATE

Value	Name	Description
0		6-tap filter (regular filter mode)
1		2-tap bilinear filter (simple profile/version mode)

3 31:30 **Reserved**

Access:	RO
Format:	MBZ

29:24 **DBLKFilterLevel for Segment3**

Exists If:	//Decoder / Encoder
Format:	U6

Value	Name	Description
0	Signifies disable in loop deblocking operation	This is used to set a VP8 profile without in loop deblocker.

Programming Notes
There are max 4 segments per frame, each segment can have its own deblocking filter level. When segmentation is disabled, only segment 0 parameter is used for the entire frame.

23:22 **Reserved**

Access:	RO
Format:	MBZ

21:16 **DBLKFilterLevel for Segment2**

Exists If:	//Decoder / Encoder
Format:	U6

Value	Name	Description
0	Signifies disable in loop deblocking operation	This is used to set a VP8 profile without in loop deblocker.

Programming Notes
There are max 4 segments per frame, each segment can have its own deblocking filter level. When segmentation is disabled, only segment 0 parameter is used for the entire frame.

15:14 **Reserved**

Access:	RO
Format:	MBZ

13:8 **DBLKFilterLevel for Segment1**

Exists If:	//Decoder / Encoder
Format:	U6

<table>
<tr><th colspan="3">MFX_VP8_PIC_STATE</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Signifies disable in loop deblocking operation</td><td>This is used to set a VP8 profile without in loop deblocker.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>There are max 4 segments per frame, each segment can have its own deblocking filter level. When segmentation is disabled, only segment 0 parameter is used for the entire frame.</td></tr>
</table>
</td></tr>
<tr><td></td><td>7:6</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>5:0</td><td><b>DBLKFilterLevel for Segment0</b>
<table>
<tr><td>Exists If:</td><td>//Decoder / Encoder</td></tr>
<tr><td>Format:</td><td>U6</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Signifies disable in loop deblocking operation</td><td>This is used to set a VP8 profile without in loop deblocker.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>There are max 4 segments per frame, each segment can have its own deblocking filter level. When segmentation is disabled, only segment 0 parameter is used for the entire frame.</td></tr>
</table>
</td></tr>
<tr><td>4</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>30:24</td><td><b>Seg 3 Qindex</b>
<table>
<tr><td>Exists If:</td><td>//Encoder Only</td></tr>
<tr><td>Format:</td><td>U7</td></tr>
</table>
Quantizer Value for Segment ID 3</td></tr>
<tr><td></td><td>23</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>22:16</td><td><b>Seg 2 Qindex</b>
<table>
<tr><td>Exists If:</td><td>//Encoder Only</td></tr>
<tr><td>Format:</td><td>U7</td></tr>
</table>
Quantizer Value for Segment ID 2</td></tr>
<tr><td></td><td>15</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
</table>

MFX_VP8_PIC_STATE

	14:8	**Seg 1 Qindex** Exists If: //Encoder Only Format: U7 Quantizer Value for Segment ID 1
	7	**Reserved** Access: RO Format: MBZ
	6:0	**Seg 0 Qindex** Exists If: //Encoder Only Format: U7 Quantizer Value for Segment ID 0. **Programming Notes** This is the **[Default]** Qindex
5	31:29	**Reserved** Access: RO Format: MBZ
	28	**UVac Qindex Delta Sign** Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for UVac
	27:24	**UVac QindexDelta** Exists If: //Encoder Only Format: U4 Absolute Quantization index delta for UVac
	23:21	**Reserved** Access: RO Format: MBZ
	20	**UVdc Qindex Delta Sign** Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for UVdc
	19:16	**UVdc Qindex Delta** Exists If: //Encoder Only Format: U4 Absolute Quantization index delta for UVdc

MFX_VP8_PIC_STATE

	15:13	**Reserved** Access: RO Format: MBZ
	12	**Y2ac Qindex Sign** Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for Y2ac
	11:8	**Y2ac Qindex Delta** Exists If: //Encoder Only Format: U4 Absolute Quantization index delta for Y2ac
	7:5	**Reserved** Access: RO Format: MBZ
	4	**Y2ac Qindex Delta Sign** Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for Y2dc This is the **[Default]** Qindex Delta Sign
	3:0	**Y2dc Qindex Delta** Exists If: //Encoder Only Format: U4 Absolute Quantization index delta for Y2dc This is the **[Default]** Qindex Delta
6	31:5	**Reserved** Access: RO Format: MBZ
	4	**Y1dc Qindex Delta Sign** Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for Y1dc This is the **[Default]**Qindex Delta Sign
	3:0	**Y1dc Qindex Delta** Exists If: //Encoder Only

MFX_VP8_PIC_STATE		
		Absolute Quantization index delta for Y1dc This is the **[Default]**Qindex Delta
7	31:15	**Reserved** Access: RO Format: MBZ
	14:8	**Clamp Qindex high** Exists If: //Encoder Only Format: U7 Maximum Clamp Value for Qindex used in quantization.
	7	**Reserved** Access: RO Format: MBZ
	6:0	**Clamp Qindex Low** Exists If: //Encoder Only Format: U7 Minimum Clamp Value for Qindex used in quantization.
8	31:25	**Reserved** Access: RO Format: MBZ
	24:16	**Quantizer Value [1][BlockType3=UVAC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Access: RO Format: MBZ
	8:0	**Quantizer Value [1][BlockType2=UVDC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
9	31:25	**Reserved** Access: RO Format: MBZ
	24:16	**Quantizer Value [1][BlockType5=Y2AC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**

MFX_VP8_PIC_STATE

	15:9	**Reserved**	
		Access:	RO
		Format:	MBZ
	8:0	**Quantizer Value [1][BlockType4=Y2DC]**	
		Exists If:	//Decoder Only
		Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]	
10	31:25	**Reserved**	
		Access:	RO
		Format:	MBZ
	24:16	**Quantizer Value [2][BlockType1=Y1AC]**	
		Exists If:	//Decoder Only
		Format:	U9
		Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]	
	15:9	**Reserved**	
		Access:	RO
		Format:	MBZ
	8:0	**Quantizer Value [2][BlockType0=Y1DC]**	
		Exists If:	//Decoder Only
		Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]	
11	31:25	**Reserved**	
		Access:	RO
		Format:	MBZ
	24:16	**Quantizer Value [2][BlockType3=UVAC]**	
		Exists If:	//Decoder Only
		Format:	U9
		Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]	
	15:9	**Reserved**	
		Access:	RO
		Format:	MBZ
	8:0	**Quantizer Value [2][BlockType2=UVDC]**	
		Exists If:	//Decoder Only
		Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]	
12	31:25	**Reserved**	
		Access:	RO
		Format:	MBZ

MFX_VP8_PIC_STATE		
	24:16	**Quantizer Value [2][BlockType5=Y2AC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Access: RO Format: MBZ
	8:0	**Quantizer Value [2][BlockType4=Y2DC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
13	31:25	**Reserved** Access: RO Format: MBZ
	24:16	**Quantizer Value [3][BlockType1=Y1AC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Access: RO Format: MBZ
	8:0	**Quantizer Value [3][BlockType0=Y1DC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
14	31:25	**Reserved** Access: RO Format: MBZ
	24:16	**Quantizer Value [3][BlockType3=UVAC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Access: RO Format: MBZ
	8:0	**Quantizer Value [3][BlockType2=UVDC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**

<table>
<tr><th colspan="3">MFX_VP8_PIC_STATE</th></tr>
<tr><td>15</td><td>31:25</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>24:16</td><td><b>Quantizer Value [3][BlockType5=Y2AC]</b>
<table><tr><td>Exists If:</td><td>//Decoder Only</td></tr><tr><td>Format:</td><td>U9</td></tr></table>
<b>Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]</b></td></tr>
<tr><td></td><td>15:9</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>8:0</td><td><b>Quantizer Value [3][BlockType4=Y2DC]</b>
<table><tr><td>Exists If:</td><td>//Decoder Only</td></tr></table>
<b>Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]</b></td></tr>
<tr><td>16..17</td><td>63:0</td><td><b>CoeffProbability StreamIn Base Address</b>
<table><tr><td>Exists If:</td><td>//Decoder Only</td></tr><tr><td>Format:</td><td><b>SplitBaseAddress4KByteAligned</b></td></tr></table>
It is specified for non-key frame only. It is the final computed probability table for parsing Coeff in the bitstream. The buffer is unsigned 8-bit * 1056 entries (CoeffProbs[4][8][3][11].</td></tr>
<tr><td>18</td><td>31:15</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:13</td><td><b>CoeffProbability StreamIn - Tiled Resource Mode</b>
<table><tr><td>Exists If:</td><td>//Decoder Only</td></tr><tr><td>Format:</td><td>U2</td></tr></table>
<b>For Media Surfaces:</b> This field specifies the tiled resource mode.
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>TRMODE_NONE</td><td>No tiled resource</td></tr>
<tr><td>1h</td><td>TRMODE_TILEYF</td><td>4KB tiled resources</td></tr>
<tr><td>2h</td><td>TRMODE_TILEYS</td><td>64KB tiled resources</td></tr>
<tr><td>3h</td><td>Reserved</td><td></td></tr></table></td></tr>
<tr><td></td><td>12:11</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>CoeffProbability StreamIn - Memory Compression Mode</b>
<table><tr><td>Exists If:</td><td>//Decoder Only</td></tr><tr><td>Format:</td><td>U1</td></tr></table>
Distinguishes Vertical from Horizontal compression. Please refer to <b>Memory Data Formats</b>, <b>Media Memory Compression</b> for more details.</td></tr>
</table>

MFX_VP8_PIC_STATE

<table>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Horizontal Compression Mode</td></tr><tr><td>1</td><td>Vertical Compression Mode</td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>CoeffProbability StreamIn - Memory Compression Enable</b><table><tr><td>Exists If:</td><td>//Decoder Only</td></tr><tr><td>Format:</td><td>Enable</td></tr></table>Memory compression will be attempted for this surface.</td></tr>
<tr><td></td><td>8:7</td><td><b>CoeffProbability StreamIn - Arbitration Priority Control</b><table><tr><td>Exists If:</td><td>//Decoder Only</td></tr><tr><td>Format:</td><td>U2</td></tr></table>This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>Highest priority</td></tr><tr><td>01b</td><td>Second highest priority</td></tr><tr><td>10b</td><td>Third highest priority</td></tr><tr><td>11b</td><td>Lowest priority</td></tr></table></td></tr>
<tr><td></td><td>6:1</td><td><b>CoeffProbability StreamIn Address - Index to Memory Object Control State (MOCS) Tables</b><table><tr><td>Exists If:</td><td>//Encoder Only</td></tr><tr><td>Format:</td><td>U6</td></tr></table>The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.</td></tr>
<tr><td></td><td>0</td><td><b>Reserved</b></td></tr>
<tr><td>19</td><td>31:24</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>MBSegmentIDTreeProbs[2]</b><table><tr><td>Exists If:</td><td>//Decoder / Encoder</td></tr><tr><td>Format:</td><td>U8</td></tr></table>MBSegmentIDTreeProbs[2:0] probability tree table for CPBAC parsing Segment_ID of each MB.</td></tr>
<tr><td></td><td>15:8</td><td><b>MBSegmentIDTreeProbs[1]</b><table><tr><td>Exists If:</td><td>//Decoder / Encoder</td></tr><tr><td>Format:</td><td>U8</td></tr></table>MBSegmentIDTreeProbs[2:0] probability tree table for CPBAC parsing Segment_ID of each MB.</td></tr>
</table>

MFX_VP8_PIC_STATE

	7:0	**MBSegmentIDTreeProbs[0]** Exists If: //Decoder / Encoder Format: U8 MBSegmentIDTreeProbs[2:0] probability tree table for CPBAC parsing Segment_ID of each MB.
20	31:24	**MBNoCoeffSkipFalseProb** Exists If: //Decoder / Encoder Format: U8 8-bit probability value for CPBAC parsing of the MBNoCoeffSkip Flag in the bistream.
	23:16	**IntraMBProb** Exists If: //Decoder / Encoder Format: U8 8-bit probability value for CPBAC parsing of the intra or inter MB type flag in the bitstream.
	15:8	**InterPredFromLastRefProb** Exists If: //Decoder / Encoder Format: U8 8-bit probability value for CPBAC parsing of the flag in the bitstream that determines which reference frame to be used for the current MB motion compensation.
	7:0	**InterPredFromGRefRefProb** Exists If: //Decoder / Encoder Format: U8 8-bit probability value for CPBAC parsing of the flag in the bitstream that determines which reference frame to be used for the current MB motion compensation.
21	31:24	**YModeProb[3]** Exists If: //Decoder / Encoder Format: U8 YModeProb[3:0] probability tree table for CPBAC parsing Luma MBType of each MB.
	23:16	**YModeProb[2]** Exists If: //Decoder / Encoder Format: U8 YModeProb[3:0] probability tree table for CPBAC parsing Luma MBType of each MB.
	15:8	**YModeProb[1]** Exists If: //Decoder / Encoder Format: U8 YModeProb[3:0] probability tree table for CPBAC parsing Luma MBType of each MB.
	7:0	**YModeProb[0]** Exists If: //Decoder / Encoder Format: U8 YModeProb[3:0] probability tree table for CPBAC parsing Luma MBType of each MB.

MFX_VP8_PIC_STATE		
22	31:24	**Reserved** Access: RO Format: MBZ
	23:16	**UVModeProb[2]** Exists If: //Decoder / Encoder Format: U8 UVModeProb[2:0] probability tree table for CPBAC parsing Chroma MBType of each MB.
	15:8	**UVModeProb[1]** Exists If: //Decoder / Encoder Format: U8 UVModeProb[2:0] probability tree table for CPBAC parsing Chroma MBType of each MB.
	7:0	**UVModeProb[0]** Exists If: //Decoder / Encoder Format: U8 UVModeProb[2:0] probability tree table for CPBAC parsing Chroma MBType of each MB.
23	31:24	**MVUpdateProbs[0][3]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[0][2]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[0][1]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[0][0]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
24	31:24	**MVUpdateProbs[0][7]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB.

MFX_VP8_PIC_STATE		
		To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[0][6]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[0][5]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[0][4]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
25	31:24	**MVUpdateProbs[0][11]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB.
	23:16	**MVUpdateProbs[0][10]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB.
	15:8	**MVUpdateProbs[0][9]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB.
	7:0	**MVUpdateProbs[0][8]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB.
26	31:24	**MVUpdateProbs[0][15]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].

MFX_VP8_PIC_STATE

	23:16	**MVUpdateProbs[0][14]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[0][13]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[0][12]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
27	31:24	**Reserved** Access: | RO Format: | MBZ
	23:16	**MVUpdateProbs[0][18]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[0][17]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[0][16]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
28	31:24	**MVUpdateProbs[1][3]** Exists If: | //Decoder Only Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].

MFX_VP8_PIC_STATE

	23:16	**MVUpdateProbs[1][2]** Exists If: | //Decoder Only Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[1][1]** Exists If: | //Decoder Only Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][0]** Exists If: | //Decoder Only Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
29	31:24	**MVUpdateProbs[1][7]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[1][6]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[1][5]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][4]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
30	31:24	**MVUpdateProbs[1][11]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].

MFX_VP8_PIC_STATE

	23:16	**MVUpdateProbs[1][10]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[1][9]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][8]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
31	31:24	**MVUpdateProbs[1][15]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[1][14]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[1][13]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][12]** Exists If: | //Decoder / Encoder Format: | U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
32	31:24	**Reserved** Access: | RO Format: | MBZ

<table>
<tr><th colspan="3">MFX_VP8_PIC_STATE</th></tr>
<tr><td></td><td>23:16</td><td><b>MVUpdateProbs[1][18]</b><table><tr><td>Exists If:</td><td>//Decoder / Encoder</td></tr><tr><td>Format:</td><td>U8</td></tr></table>MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].</td></tr>
<tr><td></td><td>15:8</td><td><b>MVUpdateProbs[1][17]</b><table><tr><td>Exists If:</td><td>//Decoder / Encoder</td></tr><tr><td>Format:</td><td>U8</td></tr></table>MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].</td></tr>
<tr><td></td><td>7:0</td><td><b>MVUpdateProbs[1][16]</b><table><tr><td>Exists If:</td><td>//Decoder / Encoder</td></tr><tr><td>Format:</td><td>U8</td></tr></table>MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].</td></tr>
<tr><td>33</td><td>31</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>30:24</td><td><b>RefLFDelta3 (for ALTREF FRAME)</b><table><tr><td>Exists If:</td><td>//Decoder / Encoder</td></tr><tr><td>Format:</td><td>S6</td></tr></table><table><tr><td>Delta value for reference frame-based adjustment of the MB-level's filter level value.</td></tr><tr><td>RefLFDeltas [ref_frametype = 0 to 3]</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>Please note that although RefDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.</td></tr></table></td></tr>
<tr><td></td><td>23</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>22:16</td><td><b>RefLFDelta2 (for GOLDEN FRAME)</b><table><tr><td>Exists If:</td><td>//Decoder / Encoder</td></tr><tr><td>Format:</td><td>S6</td></tr></table><table><tr><td>Delta value for reference frame based adjustment of the MB-level's filter level value.</td></tr><tr><td>RefLFDeltas [ref_frametype = 0 to 3]</td></tr></table></td></tr>
</table>

MFX_VP8_PIC_STATE

		Programming Notes Please note that although RefDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.
	15	**Reserved** Access: RO Format: MBZ
	14:8	**RefLFDelta1 (for LAST FRAME)** Exists If: //Decoder / Encoder Format: S6 Delta value for reference frame based adjustment of the MB-level's filter level value. RefLFDeltas [ref_frametype = 0 to 3] **Programming Notes** Please note that although RefDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.
	7	**Reserved** Access: RO Format: MBZ
	6:0	**RefLFDelta0 (for INTRA FRAME)** Exists If: //Decoder / Encoder Format: S6 Delta value for reference frame based adjustment of the MB-level's filter level value. RefLFDeltas [ref_frametype = 0 to 3] **Programming Notes** Please note that although RefDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.
34	31	**Reserved** Access: RO Format: MBZ
	30:24	**ModeLFDelta3 (for SPLITMV mode)** Exists If: //Decoder / Encoder Format: S6 Delta value for mode based adjustment of the MB-level's filter level value. ModeLFDeltas[MB_Type = 0 to 3]

MFX_VP8_PIC_STATE

Programming Notes
Please note that although ModeLFDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.

23 **Reserved**

Access:	RO
Format:	MBZ

22:16 **ModeLFDelta2 (for Nearest, Near and New mode)**

Exists If:	//Decoder / Encoder
Format:	S6

Delta value for mode based adjustment of the MB-level's filter level value.
ModeLFDeltas[MB_Type = 0 to 3]

Programming Notes
Please note that although ModeLFDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.

15 **Reserved**

Access:	RO
Format:	MBZ

14:8 **ModeLFDelta1(for ZEROMV mode)**

Exists If:	//Decoder / Encoder
Format:	S6

Delta value for mode based adjustment of the MB-level's filter level value.
ModeLFDeltas[MB_Type = 0 to 3]

Programming Notes
Please note that although ModeLFDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.

7 **Reserved**

Access:	RO
Format:	MBZ

6:0 **ModeLFDelta0 (for B_PRED mode)**

Exists If:	//Decoder / Encoder
Format:	S6

Delta value for mode based adjustment of the MB-level's filter level value.

<table>
<tr><th colspan="3">MFX_VP8_PIC_STATE</th></tr>
<tr><td></td><td></td><td>
<table><tr><td>ModeLFDeltas[MB_Type = 0 to 3]</td></tr></table>
<table><tr><th>Programming Notes</th></tr><tr><td>Please note that although ModeLFDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.</td></tr></table>
</td></tr>
<tr><td>35..36</td><td>63:0</td><td>
<b>Segmentation ID Stream Base Address</b>
<table><tr><td>Exists If:</td><td>//Decoder Only</td></tr><tr><td>Format:</td><td>SplitBaseAddress4KByteAligned</td></tr></table>
It is specified when <b>SegmentationIDStreamInEnable</b> or <b>SegmentationIDStreamOutEnable</b> is specified.
<table><tr><th>Programming Notes</th></tr><tr><td>Each cache has only 8 bits for 4 segmentation ID from 4 continuous MBs.</td></tr></table>
</td></tr>
<tr><td>37</td><td>31:15</td><td>
<b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table>
</td></tr>
<tr><td></td><td>14:13</td><td>
<b>Segmentation ID Stream - Tiled Resource Mode</b>
<table><tr><td>Exists If:</td><td>//Decoder Only</td></tr><tr><td>Format:</td><td>U2</td></tr></table>
<b>For Media Surfaces:</b> This field specifies the tiled resource mode.
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>TRMODE_NONE</td><td>No tiled resource</td></tr>
<tr><td>1h</td><td>TRMODE_TILEYF</td><td>4KB tiled resources</td></tr>
<tr><td>2h</td><td>TRMODE_TILEYS</td><td>64KB tiled resources</td></tr>
<tr><td>3h</td><td>Reserved</td><td></td></tr></table>
</td></tr>
<tr><td></td><td>12:11</td><td>
<b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table>
</td></tr>
<tr><td></td><td>10</td><td>
<b>Segmentation ID Stream - Memory Compression Mode</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
Distinguishes Vertical from Horizontal compression. Please refer to <b>Memory Data Formats</b>, <b>Media Memory Compression</b> for more details.
<table><tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Horizontal Compression Mode</td></tr>
<tr><td>1</td><td>Vertical Compression Mode</td></tr></table>
</td></tr>
<tr><td></td><td>9</td><td>
<b>Segmentation ID Stream - Memory Compression Enable</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
Memory compression will be attempted for this surface.
</td></tr>
</table>

<table>
<tr><th colspan="3">MFX_VP8_PIC_STATE</th></tr>
<tr><td></td><td>8:7</td><td><b>Segmentation ID Stream - Arbitration Priority Control</b>
<table>
<tr><td>Exists If:</td><td>//Decoder Only</td></tr>
<tr><td>Format:</td><td>U2</td></tr>
</table>
This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>Highest priority</td></tr>
<tr><td>01b</td><td>Second highest priority</td></tr>
<tr><td>10b</td><td>Third highest priority</td></tr>
<tr><td>11b</td><td>Lowest priority</td></tr>
</table>
</td></tr>
<tr><td></td><td>6:1</td><td><b>CoeffProbability StreamIn Address - Index to Memory Object Control State (MOCS) Tables</b>
<table>
<tr><td>Format:</td><td>U6</td></tr>
</table>
The index to define the L3 and system cache memory properties. The details of the controls are further defined in L3 and Page walker (memory interface) control registers. The field is defined to populate 64 different surface controls to be used concurrently. Related control registers can be updated during runtime.
</td></tr>
<tr><td></td><td>0</td><td><b>Reserved</b></td></tr>
</table>

MFX_WAIT

MFX_WAIT

Source:	VideoCS
Length Bias:	1

This command can be considered the same as an MI_NOOP except that the command parser will not parse the next command until the following happens

- **AVC or VC1 BSD mode:** The command will stall the parser until completion of the BSD object
- **IT, encoder, and MPEG2 BSD mode:** The command will stall the parser until the object package is sent down the pipeline This command should be used to ensure the preemption enable window occurs during the time the object command is being executed down the pipeline.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 03h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Command Subtype** Default Value: 01h MFX_SINGLE_DW Format: OpCode
	26:16	**Sub-Opcode** Default Value: 0h MFX_WAIT Format: OpCode
	15:10	**Reserved** Access: RO Format: MBZ
	9	**Reserved**
	8	**MFX Sync Control Flag**
	7:6	**Reserved** Access: RO Format: MBZ
	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2

MI_ARB_CHECK

MI_ARB_CHECK

Source:	CommandStreamer
Length Bias:	1

Description
This command allows software to enable or disable pre-fetch mechanism for command buffers in hardware.
The command allows software to add preemption points in the ring buffer. The command streamer will preempt in the case arbitration is enabled, there is a pending execution list and this command is currently being parsed.

Programming Notes
MI_ARB_CHK command can be programmed in a ring buffer or batch buffer. MI_ARB_CHK command must not be programmed in INDIRECT_CTX and BB_PER_CTX_PTR buffers.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table><tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b>
<table><tr><td>Default Value:</td><td>05h MI_ARB_CHECK</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>22:16</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:8</td><td><b>Mask Bits</b>
<table><tr><th>Programming Notes</th></tr><tr><td>Must be set to modify corresponding bits in Bits 7:0. (For implemented bits)</td></tr></table></td></tr>
<tr><td></td><td>7:1</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>Pre-Parser Disable</b>
This command allows software to enable or disable pre-parser of command buffer functionality from within a command sequence on per context basis. This ability allows the command stream to prefetch batch buffers that are yet to be parsed by looking ahead in the command FIFO. Even with this disabled, driver will have to ensure it does not self-modify commands already prefetched into the command buffer within the same batch buffer.
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>1</td><td></td><td>When set early fetch and parsing of future command buffers is disabled in hardware.</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MI_ARB_CHECK</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>0</td><td></td><td>When reset early fetch and parsing of future command buffers is enabled in hardware when "Pre-Fetch Disable" in GFX_MODE register is not set.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Mask bit [8] must be set to modify this bit. By default pre-fetch in hardware is enabled. The status of this bit is engine context save/restored.</td></tr>
<tr><td>This is not a preemptible command if the corresponding mask bit is set for this field.</td></tr>
</table>
</td></tr>
</table>

MI_ARB_ON_OFF

MI_ARB_ON_OFF

Source:	CommandStreamer
Length Bias:	1

The MI_ARB_ON_OFF instruction is used to disable/enable context switching. This instruction can be used to prevent submission of a new execlist from interrupting a command sequence, however lite restore preemption is allowed with in the arbitration disabled command execution zone. Note that context switching will remain disabled until re-enabled through use of this command. This command will also prevent a switch in the case of waiting on events, running out of commands. These will effectively hang the device if allowed to occur while arbitration is off (context switching is disabled.) This command should always be used as an off-on pair with the sequence of instructions to be protected from context switch between MI_ARB_OFF and MI_ARB_ON. Software must use this arbitration control with caution since it has the potential to increase the response time of the Render Engine to pre-emption requests. This is a privileged command; it will not be effective (will be converted to a no-op) if executed from within a non-privileged batch buffer.

The MI_ARB_ON_OFF instruction is used to disable/enable context switching. Context switching could be either due to preemption or un-successful wait for events or semaphore waits. This instruction can be used to prevent submission of a new execlist from interrupting a command sequence, however lite restore preemption is allowed with in the arbitration disabled command execution zone. Note that context switching will remain disabled until re-enabled through use of this command. This command will also prevent a switch in the case of waiting on events, running out of commands. These will effectively hang the device if allowed to occur while arbitration is off (context switching is disabled.)

Programming Notes

This command must always be programmed in pairs of off/on in the same command dispatch. Sequence of instructions to be protected from context switch or preemption must be programmed between the MI_ARB_OFF and MI_ARB_ON. Software must use this arbitration control with caution since it has the potential to increase the response time of the Render Engine to pre-emption requests. This is a privileged command; it will not be effective (will be converted to a no-op) if executed from within a non-privileged batch buffer.

MI_ARB_ON_OFF command must not be programmed as part of the POSH command execution.

DWord	Bit	Description	
0	31:29	**Command Type**	
		Default Value:	0h MI_COMMAND
		Format:	OpCode
	28:23	**MI Command Opcode**	
		Default Value:	08h MI_ARB_ON_OFF
		Format:	OpCode
	22:2	**Reserved**	
		Access:	RO
		Format:	MBZ

MI_ARB_ON_OFF

1 **Arbitration Mode**

Source:	RenderCS
Format:	Enable

This bit controls whether or not lite restore is allowed when arbitration is disabled thru clearing the Arbitration Enable bit. If arbitration is enabled, then the value of this bit does not change the behavior of the hardware.

Value	Name
0h	Allow Lite Restore **[Default]**
1h	Lite Restore Disabled

0 **Arbitration Enable**

Format:	Enable

This field enables or disables context switches due to pre-emption (a new execlist).

Value	Name
1	Arbitration Enabled **[Default]**
0	Arbitration Disabled

MI_ATOMIC

MI_ATOMIC

Source:	BSpec
Length Bias:	2

Description

MI_ATOMIC is used to carry atomic operation on data in graphics memory. Atomic operations are supported on data granularity of 4B, 8B and 16B. The atomic operation leads to a read-modify-write operation on the data in graphics memory with the option of returning value. The data in graphics memory is modified by doing arithmetic and logical operation with the inline/indirect data provided with the MI_ATOMIC command.
Inline/Indirect provided in the command can be one or two operands based on the atomic operation. Ex: Atomic-Compare operation needs two operands while Atomic-Add operation needs single operand and Atomic-increment requires no operand. Refer "Atomics" sub-section of "L3 Cache and URB" section of the B-spec for detailed atomic operations supported. Atomic operations can be enabled to return value by setting "Return Data Control" field in the command, return data is stored to CS_GPR registers.
CS_GPR4/5 registers are updated with memory Return Data based on the "Data Size". Each GPR register is qword in size and occupies two MMIO registers.
Note: Any references to CS_GPR registers in the command should be understood as the CS_GPR registers belonging to the corresponding engines *CS_GPR registers.

Engine Name	Corresponding GPR Registers
RCS, POCS	CS_GPR, POCS_GPR
BCS	BCS_GPR
VCS	VCS_GPR
VECS	VECS_GPR

Indirect Source Operands:
Operand1 is sourced from [CS_GPR1, CS_GPR0]
Operand2 is sourced from [CS_GPR3, CS_GPR2]
Read return Data is stored in [CS_GPR_5, CS_GPR4]

When "Data Size" is DWORD lower dword of CS_GPR4 (Qword) is updated with the dword data returned from memory. When "Data Size" is QWORD only CS_GPR4 (Qword) is updated with the qword data returned from memory. When the data size is OCTWORD CS_GPR4/5 are updated with the OCTWORD data returned from memory. CS_GPR4 is loaded with lower qword returned from memory and CS_GPR5 is loaded with upper qword returned from memory.

Programming Notes

- When Inline Data mode is not set, Dwords 3..10 must not be included as part of the command. Dword Length field in the header must be programmed accordingly.
- When Inline Data Mode is set, Dwords3..10 must be included based on the Data Size field of the header. Both Operand-1 and Operand-2 dwords must be programmed based on the Data Size field. Operand-2 must be programmed to 0x0 if the atomic operation doesn't require it. Dword Length field in the header must be programmed accordingly.

DWord	Bit	Description

MI_ATOMIC

<table>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table>
<tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table>
</td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b>
<table>
<tr><td>Default Value:</td><td>2Fh MI_ATOMIC</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table>
</td></tr>
<tr><td></td><td>22</td><td><b>Memory Type</b>
This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit must be 1 if the <b>Per Process GTT Enable</b> bit is clear.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Per Process Graphics Address</td><td></td></tr>
<tr><td>1h</td><td>Global Graphics Address</td><td>This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.</td></tr>
</table>
</td></tr>
<tr><td></td><td>21</td><td><b>Post-Sync Operation</b>
<table>
<tr><td>Source:</td><td>RenderCS, PositionCS</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>No Post Sync Operation</td><td>Command is executed as usual.</td></tr>
<tr><td>1h</td><td>Post Sync Operation</td><td>MI_ATOMIC command is executed as a pipelined PIPE_CONTROL flush command with Atomics operation as post sync operation. Flush completion only guarantees the workload prior to this command is pushed till Windower unit and completion of any outstanding flushes issued prior to this command.
When this bit set following restriction apply to atomic operation:
<ul>
<li>Non-Compare atomic operations are supported on data granularity of 4B and 8B. DW3 is the lower dword of the operand and DW4 is the upper dword of the operand for the atomic operation.</li>
<li>Compare atomic operations are supported on data granularity of 4B. DW3 is Operand-0 and DW4 is Operand-1 for the atomic operation.</li>
<li>Atomic operations to GGTT/PPGTT memory surface are supported.</li>
<li>Only Inline data mode for atomic operand is supported, no support for indirect data mode.</li>
<li>No support for Return Data Control functionality.</li>
<li>No support for atomic operations on data granularity of 16B.</li>
</ul>
</td></tr>
</table>
</td></tr>
</table>

MI_ATOMIC

		• No support for compare atomic operations on data granularity of 8B.

Programming Notes
Any desired pipeline flush operation can be achieved by programming PIPE_CONTROL command prior to this command.
When this bit is set Command Streamer sends a flush down the pipe and the atomic operation is saved as post sync operation. Command streamer goes on executing the following commands. Atomic operation saved as post sync operation is executed at some point later on completion of corresponding flush issued.
When this bit is set atomic semaphore signal operation will be out of order with rest of the MI commands programmed in the ring buffer or batch buffer, it will be in order with respect to the post sync operations resulting due to PIPE_CONTROL command.

20:19 **Data Size**

This field indicates the size of the operand in dword/qword/octword on which atomic operation will be performed. Data size must match with the Atomic Opcode. Operation Data size could be 4B, 8B or 16B

Value	Name	Description
0h	DWORD	Operand size used by Atomic Operation is DWORD.
1h	QWORD	Operand Size used by Atomic Operation is QWORD.
2h	OCTWORD	Operand Size used by Atomic Operation is OCTWORD.
3h	RESERVED	

18 **Inline Data**

This bit when set indicates the source operands are provided in line within the command. When reset the source operands are in CS_GPR registers.

Programming Notes
CS_GPR registers must be programmed with appropriate values before issuing MI_ATOMIC command with this field reset.

17 **CS STALL**

This bit when set command stream waits for completion of this command before executing the next command.

Programming Notes
Render Command Streamer Only: CS will not guarantee atomic operation to be complete upon setting this bit along with Post Sync Operation set. When Post Sync Operation is set, this bit has no significance.

16 **Return Data Control**

Source:	RenderCS, BlitterCS, VideoCS, VideoEnhancementCS

When Return Data Control is set the read return feature will be enabled during the atomic operation. Data is stored in CS_GPR5/4 registers unconditionally on completion of the atomic operation. On data return CS_GPR5/4 Registers are updated based on the "Data Size" field. When

MI_ATOMIC

		"Data Size" is DWORD lower dword of CS_GPR4 (Qword) is updated with the dword data returned from memory. When "Data Size" is QWORD only CS_GPR4 (Qword) is updated with the qword data returned from memory. When the data size is OCTWORD CS_GPR4/5 are updated with the OCTWORD data returned from memory. CS_GPR4 is loaded with lower qword returned from memory and CS_GPR5 is loaded with upper qword returned from memory.
	15:8	**ATOMIC OPCODE** This field selects the kind of atomic operation to be performed. Refer "Atomics" sub-section of "L3 Cache and URB" section for atomic opcode encoding and operation.

Value	Name
1h	Atomic_AND
2h	Atomic_OR
3h	Atomic_XOR
4h	Atomic_MOVE
5h	Atomic_INC
6h	Atomic_DEC
7h	Atomic_ADD
8h	Atomic_SUB
87h	Atomic_FADD
88h	Atomic_FSUB
Ah	Atomic_IMAX
Bh	Atomic_IMIN
Ch	Atomic_UMAX
Dh	Atomic_UMIN
Eh	Atomic_CMP/WR
9Bh	Atomic_Min_Float16
9Ah	Atomic_Max_Float16
9Eh	Atomic_FLOATCMP/WR16
21h	Atomic_AND8B
22h	Atomic_OR8B
23h	Atomic_XOR8B
24h	Atomic_MOVE8B
25h	Atomic_INC8B
26h	Atomic_DEC8B
27h	Atomic_ADD8B
28h	Atomic_SUB8B
2Ah	Atomic_IMAX8B
2Bh	Atomic_IMIN8B
2Ch	Atomic_UMAX8B

MI_ATOMIC

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>2Dh</td><td>Atomic_UMIN8B</td></tr>
<tr><td>2Eh</td><td>Atomic_CMP/WR8B</td></tr>
<tr><td>8Ah</td><td>Atomic_MAX_Float32</td></tr>
<tr><td>8Bh</td><td>Atomic_MIN_Float32</td></tr>
<tr><td>8Eh</td><td>Atomic_CMP/WR_Float32</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>The following opcodes are supported for atomics towards a system memory type:<ul><li>Atomic_MOVE</li><li>Atomic_INC</li><li>Atomic_DEC</li></ul></td></tr>
</table>
</td></tr>
<tr><td></td><td>7:0</td><td>
<b>DWord Length</b>
<table>
<tr><td>Format:</td><td>=n</td></tr>
</table>
Total Length - 2. Excludes DWord (0,1).
<table>
<tr><th>Value</th><th>Name</th><th>Exists If</th></tr>
<tr><td>1h</td><td>Inline Data 0 <b>[Default]</b></td><td>([Inline Data]==0)</td></tr>
<tr><td>9h</td><td>Inline Data 1</td><td>([Inline Data]==1)</td></tr>
</table>
</td></tr>
<tr><td>1</td><td>31:2</td><td>
<b>Memory Address</b>
<table>
<tr><td>Format:</td><td>GraphicsAddress[31:2]</td></tr>
</table>
This field contains the graphics memory address of the data on which atomic operation has to be performed. Atomic operation can be performed on data granularity of 4B, 8B or 16B and hence the Address has to be correspondingly aligned to 4B,8B or 16B respectively.
</td></tr>
<tr><td></td><td>1</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>0</td><td>
<b>Workload Partition ID Offset Enable</b>
This bit controls the memory write address computation for the atomic operation. The final memory write address is computed by adding the Workload PartitionID times the Address Offset to the memory address mentioned in the command. Workload Partition ID gets programmed through WPARID register and the Address Offset gets programmed through CS_MI_ADDRESS_OFFSET register..
Example: Final Memory Write Address[47:2] = (Workload Partition ID * "Address Offset") + Memory Write Address [47:2]
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td></td><td>The final memory address is computed based on the Workload Partition ID</td></tr>
<tr><td>0</td><td></td><td>There is no offset added to the memory write address.</td></tr>
</table>
</td></tr>
</table>

MI_ATOMIC		
2	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Memory Address High** This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.
3	31:0	**Operand1 Data Dword 0** Format: U32 Dword0 of Operand1 when Inline Data mode is set.
4	31:0	**Operand2 Data Dword 0** Format: U32 Dword0 of Operand2 when Inline Data mode is set.
5	31:0	**Operand1 Data Dword 1** Format: U32 Dword1 of Operand1 when Inline Data mode is set.
6	31:0	**Operand2 Data Dword 1** Format: U32 Dword1 of Operand2 when Inline Data mode is set.
7	31:0	**Operand1 Data Dword 2** Format: U32 Dword2 of Operand1 when Inline Data mode is set.
8	31:0	**Operand2 Data Dword 2** Format: U32 Dword2 of Operand2 when Inline Data mode is set.
9	31:0	**Operand1 Data Dword 3** Format: U32 Dword3 of Operand1 when Inline Data mode is set.
10	31:0	**Operand2 Data Dword 3** Format: U32 Dword3 of Operand2 when Inline Data mode is set.

MI_BATCH_BUFFER_END

MI_BATCH_BUFFER_END

Source:	CommandStreamer
Length Bias:	1

The MI_BATCH_BUFFER_END command is used to terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command.

Programming Notes	Source
SW must ensure it buffers the size of the command buffer beyond this command. The command buffer for the command streamer is 0.5KB.	BlitterCS, VideoCS, VideoEnhancementCS
SW must ensure it buffers the size of the command buffer beyond this command. The command buffer for the command streamer is 1 KB.	RenderCS, ComputeCS

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Ah MI_ BATCH_BUFFER_END Format: OpCode
	22:16	**Reserved** Access: RO Format: MBZ
	15	**Predicate Enable** This bit is used to enable predication of this command. If this bit is set and Bit 0 of the MI_SET_PREDICATE_RESULT register is set, this command is ignored. Otherwise, the command is performed normally.
	14:1	**Reserved** Access: RO Format: MBZ
	0	**End Context** Format: Enable This bit must only be set within a context image. If this command is parsed with this bit set then the engine will consider the context image restore complete. This bit is ignored if parsed during a batch buffer.

MI_BATCH_BUFFER_START

MI_BATCH_BUFFER_START

Source:	CommandStreamer
Length Bias:	2

The MI_BATCH_BUFFER_START command is used to initiate the execution of commands stored in a batch buffer. For restrictions on the location of batch buffers, see Batch Buffers in the Device Programming Interface chapter of *MI Functions*. The batch buffer can be specified as privileged or non-privileged, determining the operations considered valid when initiated from within the buffer and any attached (chained) batch buffers. See Batch Buffer Protection in the Device Programming Interface chapter of *MI Functions*.

Programming Notes	Source
• A batch buffer initiated with this command must end either with a MI_BATCH_BUFFER_END command or by chaining to another batch buffer with an MI_BATCH_BUFFER_START command. • It is essential that the address location beyond the current page be populated inside the GTT. HW performs over-fetch of the command addresses and any over-fetch requires a valid TLB entry. A single extra page beyond the batch buffer is sufficient. Otherwise the driver can check if the current page has enough space for the size of the overfetch.	
SW must ensure it buffers the size of the batch buffer includes additional buffer equal to the command buffer beyond the end. The command buffer for the command streamer is 0.5KB.	BlitterCS, VideoCS, VideoEnhancementCS
SW must ensure it buffers the size of the batch buffer includes additional buffer equal to the command buffer beyond the end. The command buffer for the command streamer is 1 KB.	RenderCS, ComputeCS

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b><table><tr><td>Default Value:</td><td>31h MI_BATCH_BUFFER_START</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>22</td><td><b>Second Level Batch Buffer</b><table><tr><td>Exists If:</td><td>//MI_MODE:NestedBatchBufferEnable=='0'</td></tr></table>The command streamer contains three storage elements; one for the ring head address, one for the batch head address, and one for the second level batch head address. When performing batch buffer chaining, hardware simply updates the head pointer of the first level batch address storage. There is no stack in hardware. When this bit is set, hardware uses the 2nd level batch head address storage element. Chaining of second level batch buffers is supported. A chained second level batch buffer is inferred in hardware when a</td></tr>
</table>

MI_BATCH_BUFFER_START

MI_BATCH_BUFFER_START command with "Second Level Batch Buffer" bit field set is executed from a second level batch buffer, hardware simply updates the head pointer of the second level batch address storage. Upon MI_BATCH_BUFFER_END, it will automatically return to the first level batch buffer address. This allows hardware to mimic a simple 3-level stack.

Value	Name	Description
0h	First level batch	Place the batch buffer address in the 1st (traditional) level batch address storage element.
1h	Second level batch	Place the batch buffer address in the second-level batch address storage element.

Programming Notes
This field is ignored when MI_BATCH_BUFFER_START is executed from a ring. Whether this is a zero or one, the command streamer will move to the first level batch buffer.

22

Nested Level Batch Buffer

Exists If:	//MI_MODE:NestedBatchBufferEnable=='1'

Description
If this bit is set, the command streamer will move to the next level of batch buffer. Once it executes a MI_BATCH_BUFFER_END in the next level, it will return to the batch buffer executing this command and execute the next command. If clear then it will remain in the same batch buffer level and on executing MI_BATCH_BUFFER_END, it will return to the previous level. Otherwise known as batch buffer chaining. Hardware supports three levels of nesting, namely First Level, Second Level and Third Level. This bit must not be set in any of the MI_BATCH_BUFFER_START commands programmed as part of the 3rd level batch buffer's command sequence.

Value	Name	Description
0h	Chain	Stay in the same batch buffer level.
1h	Nested	Move to the next level of batch buffer.

Programming Notes
This field is ignored when MI_BATCH_BUFFER_START is executed from a ring. Whether this is a zero or one, the command streamer will move to the first level batch buffer.
Following programming guidelines must be follow for programming commands in third level batch buffer: • Preemptable commands must not be programmed inside third level batch buffer. Refer section Preemption/Execlist Scheduling (**Preemptable**

MI_BATCH_BUFFER_START

<table>
<tr><td></td><td></td><td>
<b>Commands</b>) for the list of preemptable commands supported on per engine basis. Preemptable commands for RenderCS are mentioned below.
<ul>
<li>3DPRIMTIVE and MI_ARB_CHK command in 3D mode of PIPELINE_SELECT operation.</li>
<li>GPGPU_WALKER, MEDIA_WALKER, MEDIA_OBJECT, PIPE_CONTROL, MI_ATOMIC (bit 21 set) and MEDIA_STATE_FLUSH in GPGPU or MEDIA mode of PIPELINE_SELECT operation.</li>
<li>Any Non-Pipeline state command in GPGPU mode of PIPELINE_SELECT operation.</li>
</ul>
<ul>
<li>MI_SEMAPHORE_WAIT and MI_WAIT_FOR_EVENT commands must not be programmed inside third level batch buffers. These commands are preemptable and also can result in context switch and hence must not be programmed inside third level batch buffer.</li>
</ul>
</td></tr>
<tr><td></td><td>21:20</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>19</td><td>
<b>Enable Command Cache</b>
<table>
<tr><td>Source:</td><td>RenderCS</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td></td><td>Command Cache enabled</td></tr>
<tr><td>0</td><td><b>[Default]</b></td><td>Command Cache disable</td></tr>
</table>
</td></tr>
<tr><td></td><td>18</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>17:16</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>15</td><td>
<b>Predication Enable</b>
This bit is used to enable predication of this command. If this bit is set and Bit 0 of the MI_SET_PREDICATE_RESULT register is set, this command is ignored. Otherwise the command is performed normally.
</td></tr>
<tr><td></td><td>14:10</td><td>
<b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>9</td><td><b>Reserved</b></td></tr>
</table>

<table>
<tr><th colspan="3">MI_BATCH_BUFFER_START</th></tr>
<tr><td rowspan="2"></td><td>8</td><td><b>Address Space Indicator</b>
<table>
<tr><th>Description</th></tr>
<tr><td>Batch buffers accessed via PPGTT are considered as non-privileged. Certain operations (e.g., MI_STORE_DATA_IMM commands to GGTT memory) are prohibited within non-privileged buffers. More details mentioned in User Mode Privileged command section. When MI_BATCH_BUFFER_START command is executed from within a batch buffer (i.e., is a "chained" or "second level" batch buffer command), the current active batch buffer's "Address Space Indicator" and this field determine the "Address Space Indicator" of the next buffer in the chain.
• Chained or Second level batch buffer can be in GGTT or PPGTT if the parent batch buffer is in GGTT.
• Chained or Second level batch buffer can only be in PPGTT if the parent batch buffer is in PPGTT. This is enforced by Hardware.</td></tr>
<tr><td>Batch buffers accessed via PPGTT are considered as non-privileged. Certain operations (e.g., MI_STORE_DATA_IMM commands to GGTT memory) are prohibited within non-privileged buffers. More details mentioned in User Mode Privileged command section. When MI_BATCH_BUFFER_START command is executed from within a batch buffer (i.e., is a "chained" or "second level" batch buffer command), the current active batch buffer's "Address Space Indicator" and this field determine the "Address Space Indicator" of the next buffer in the chain.
• Chained or Nested level batch buffer can be in GGTT or PPGTT if the parent batch buffer is in GGTT.
• Chained or Nested level batch buffer can only be in PPGTT if the parent batch buffer is in PPGTT. This is enforced by Hardware.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>GGTT</td><td>This batch buffer is located in GGTT memory and is privileged.</td></tr>
<tr><td>1h</td><td>PPGTT</td><td>This batch buffer is located in PPGTT memory and is Non-Privileged.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This field must be '0' unless the Per-Process GTT Enable is '1'</td></tr>
</table>
</td></tr>
<tr><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Default Value:</td><td>1h Excludes DWord (0,1)</td></tr>
<tr><td>Format:</td><td>=n</td></tr>
</table>
Total - Bias. Excludes DWord (0,1).</td></tr>
<tr><td>1..2
The address is a 64-bit</td><td>63:2</td><td><b>Batch Buffer Start Address</b>
<table>
<tr><td>Format:</td><td>VIRTUAL_ADDR[63:2]</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th colspan="3">MI_BATCH_BUFFER_START</th></tr>
<tr><td>value [63:0], but only a portion of it is used by hardware. The upper 63 down to 48 bits are reserved bits which are ignored.</td><td>1:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

MI_CONDITIONAL_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END

Source:	CommandStreamer
Length Bias:	2

Description

The MI_CONDITIONAL_BATCH_BUFFER_END command is used to conditionally terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command.
Termination of the current level of batch buffer from which MI_CONDITIONAL_BATCH_BUFFER_END is executed or termination of all levels of batch buffer behavior is controlled by the **End Current Batch Buffer Level** bit in the command header.

Programming Notes

Any updates to the memory location exercised by this command must be ensured to be coherent in memory prior to programming of this command. If the memory location is being updated by a prior executed MI command (ex: MI_STORE_REGISTER_MEM, etc.) on the same engine, SW must follow one of the below programming note prior to programming of this command to ensure data is coherent in memory.
Option1: Programming of "4" dummy MI_STORE_DATA_IMM (write to scratch space) commands prior to programming of this command. Example: MI_STORE_REGISTE_MEM (0x2288, 0x2CF0_0000)
MI_STORE_DATA_IMM (4 times) (Dummy data, Scratch Address)
MI_CONDITIONAL_BATCH_BUFFER_END(0x2CF0_0000)
Option2: Programming of a PIPE_CONTROL with Post-Sync Operation selected to Write Immediate Data to scratch space address with Command Streamer Stall Enable set prior to programming of this command. Example: MI_STORE_REGISTE_MEM (0x2288, 0x2CF0_0000) PIPE_CONTROL (Stall, Write Immediate Data), MI_CONDITIONAL_BATCH_BUFFER_END(0x2CF0_0000) .
Option3: MI_ATOMIC (write to scratch space) with "CS STALL" set prior to programming of this command. Example: MI_STORE_REGISTE_MEM (0x2288, 0x2CF0_0000) MI_ATOMIC (MOV, Dummy data, Scratch Address), MI_CONDITIONAL_BATCH_BUFFER_END(0x2CF0_0000) .

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table>
<tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b>
<table>
<tr><td>Default Value:</td><td>36h MI_CONDITIONAL_BATCH_BUFFER_END</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>22</td><td><b>Use Global GTT</b>
<table>
<tr><td>Default Value:</td><td>0h</td></tr>
<tr><td>Format:</td><td>Boolean</td></tr>
</table>
If set, this command will use the global GTT to translate the <b>Compare Address</b> and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used to translate the <b>Compare Address</b>.</td></tr>
</table>

MI_CONDITIONAL_BATCH_BUFFER_END

<table>
<tr><td></td><td>21</td><td><b>Compare Semaphore</b>
<table><tr><td>Format:</td><td>Boolean</td></tr></table>
This bit provides a means to enable or disable compare operation.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1h</td><td></td><td>The data from the Compare Data Dword (inline) is compared to the data in memory pointed by the Compare Address as per the Compare Operation. Based on the outcome of the compare operation it may result in either continue execution of the batch buffer or it may result in termination of the batch buffer.</td></tr>
<tr><td>0h</td><td></td><td>This command will be treated as NOOP and usual batch buffer execution flow continues.</td></tr>
</table></td></tr>
<tr><td></td><td>20</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>19</td><td><b>Compare Mask Mode</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Compare Mask Mode Disabled</td><td>Compare address points to Dword in memory consisting of Data Dword(DW0). HW will compare Data Dword(DW0) against Semaphore Data Dword.</td></tr>
<tr><td>1h</td><td>Compare Mask Mode Enabled</td><td>Compare address points to Qword in memory consisting of compare Mask (DW0) and Data Dword(DW1). HW will do AND operation on Mask(DW0) with Data Dword(DW1) and then compare the result against Semaphore Data Dword.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>When "Compare Mask Mode" is enabled, "Compare Address" must be qword aligned.</td></tr>
</table></td></tr>
<tr><td></td><td>18</td><td><b>End Current Batch Buffer Level</b>
This field specifies if the current level of the batch buffer execution must or the complete batch (including parent) buffer execution must be terminated on compare operation evaluating false.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td></td><td>Execution of the command results in terminating the batch buffer level from which this command has been executed and command execution returns to the previous/parent batch buffer.
Ex:<ul><li>when executed from a first level batch buffer, execution of batch buffer is terminated and the command execution resumes to the ring buffer. This is similar to as if MI_BATCH_BUFFER_END command was executed from first level batch buffer.</li><li>when executed from a second level batch buffer, execution of second level batch buffer is terminated and</li></ul></td></tr>
</table></td></tr>
</table>

MI_CONDITIONAL_BATCH_BUFFER_END

		the command execution resumes to the first level batch buffer. This is similar to as if MI_BATCH_BUFFER_END command was executed from second level batch buffer. • when executed from a third level batch buffer (if supported), execution of third level batch buffer is terminated and the command execution resumes to the second level batch buffer. This is similar to as if MI_BATCH_BUFFER_END command was executed from third level batch buffer.
0		Execution of the command result in termination of all levels of batch buffer and command execution returns to the ring buffer.

17:16 **Reserved**

Access:	RO
Format:	MBZ

15 **Predicate Enable**

This bit is used to enable predication of this command. If this bit is set and Bit 0 of the MI_SET_PREDICATE_RESULT register is set, this command is ignored. Otherwise the command is performed normally.

14:12 **Compare Operation**

This field specifies the operation that will be executed to create the result that will either allow to continue or terminate the batch buffer.

MAD = Memory Address Data Dword

IDD =Inline Data Dword

Value	Name	Description
0h	MAD_GREATER_THAN_IDD	If Indirect fetched data is greater than inline data then continue.
1h	MAD_GREATER_THAN_OR_EQUAL_IDD	If Indirect fetched data is greater than or equal to inline data then continue.
2h	MAD_LESS_THAN_IDD	If Indirect fetched data is less than inline data then continue.
3h	MAD_LESS_THAN_OR_EQUAL_IDD	If Indirect fetched data is less than or equal to inline data then continue.
4h	MAD_EQUAL_IDD	If Indirect fetched data is equal to inline data then continue.
5h	MAD_NOT_EQUAL_IDD	If Indirect fetched data is not equal to inline data then continue.
6h	Reserved	
7h	Reserved	

MI_CONDITIONAL_BATCH_BUFFER_END

	11:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n
1	31:0	**Compare Data Dword** Data dword to compare memory. The Data dword is supplied by software to control execution of the command buffer. If the compare is enabled and the data at Compare Address is greater than this dword, the execution of the command buffer should continue.
2..3 Qword address to fetch Data Qword from memory. This field specifies the 4GB aligned base address of Gfx 4GB virtual address space within the host's 64-bit virtual address space. Virtual Address is a 64-bit value [63:0], but only a portion of it is used by hardware. Upper reserved bits are ignored and MBZ.	63:3	**Compare Address** Format: VIRTUAL_ADDR[63:3]
	2:0	**Reserved** Access: RO Format: MBZ

MI_COPY_MEM_MEM

<table>
<tr><th colspan="3">MI_COPY_MEM_MEM</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">The MI_COPY_MEM_MEM command reads a DWord from memory and stores the value of that DWord to back to memory. The source and destination addresses are specified in the command. The command temporarily halts command execution.</td></tr>
<tr><td colspan="3"><table><tr><th>Programming Notes</th></tr><tr><td>This command should not be used within a "non_privilege" batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.</td></tr></table></td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b><table><tr><td>Default Value:</td><td>2Eh MI_COPY_MEM_MEM</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>22</td><td><b>Use Global GTT Source</b>
It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Per Process Graphics Address</td><td></td></tr><tr><td>1h</td><td>Global Graphics Address</td><td>This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.</td></tr></table></td></tr>
<tr><td></td><td>21</td><td><b>Use Global GTT Destination</b>
It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear.</td></tr>
</table>

MI_COPY_MEM_MEM

<table>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Per Process Graphics Address</td><td></td></tr><tr><td>1h</td><td>Global Graphics Address</td><td>This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.</td></tr></table></td></tr>
<tr><td></td><td>20:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b><table><tr><td>Default Value:</td><td>3 Excludes DWord (0,1)</td></tr><tr><td>Format:</td><td>=n</td></tr></table></td></tr>
<tr><td>1..2
Surface Type: MMIO Register
Virtual Address is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and MBZ.</td><td>63:0</td><td><b>Destination Memory Address</b><table><tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr></table></td></tr>
<tr><td>3..4
Surface Type: MMIO Register
Virtual Address is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and MBZ.</td><td>63:0</td><td><b>Source Memory Address</b><table><tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr></table></td></tr>
</table>

MI_COPY_MEM_MEM

<table>
<tr><th colspan="3">MI_COPY_MEM_MEM</th></tr>
<tr><td colspan="3">Source: RenderCS
Length Bias: 2</td></tr>
<tr><td colspan="3">The MI_COPY_MEM_MEM command reads a DWord from memory and stores the value of that DWord to back to memory. The source and destination addresses are specified in the command. The command temporarily halts command execution.</td></tr>
<tr><td colspan="3"><table><tr><th>Programming Notes</th></tr><tr><td>This command should not be used within a "non_privilege" batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.</td></tr></table></td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b><table><tr><td>Default Value:</td><td>2Eh MI_COPY_MEM_MEM</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>22</td><td><b>Use Global GTT Source</b>
It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Per Process Graphics Address</td><td></td></tr><tr><td>1h</td><td>Global Graphics Address</td><td>It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear.</td></tr></table></td></tr>
<tr><td></td><td>21</td><td><b>Use Global GTT Destination</b>
This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit <i>must</i> be '1' if the <b>Per Process GTT Enable</b> bit is clear. This bit will determine write to memory uses Global or Per Process GTT.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Per Process Graphics Address</td><td></td></tr></table></td></tr>
</table>

MI_COPY_MEM_MEM

		1h Global Graphics Address This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	20:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 3 Excludes DWord (0,1) Format: =n
1..2	63:2	**Destination Memory Address** Format: GraphicsAddress[63:2] Surface Type: MMIO Register This field specifies the address of the memory location where the value fetched specified in the DWord address above will be written. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[63:2] for a DWord register GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	1:0	**Reserved** Access: RO Format: MBZ
3..4	63:2	**Source Memory Address** Format: GraphicsAddress[63:2] Surface Type: MMIO Register This field specifies the address of the memory location where the value is located that will be written to the address below. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[63:2] for a DWord register GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	1:0	**Reserved** Access: RO Format: MBZ

MI_COPY_MEM_MEM

<table>
<tr><th colspan="3">MI_COPY_MEM_MEM</th></tr>
<tr><td colspan="3">Source: VideoCS
Length Bias: 2</td></tr>
<tr><td colspan="3">The MI_COPY_MEM_MEM command reads a DWord from memory and stores the value of that DWord to back to memory. The source and destination addresses are specified in the command. The command temporarily halts command execution.</td></tr>
<tr><td colspan="3"><table><tr><th>Programming Notes</th></tr><tr><td>This command should not be used within a "non-privileged" batch buffer to access global virtual space; doing so will be treated as privilege access violation. Refer to the "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to learn more about HW behavior on encountering a privilege access violation.
This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.</td></tr></table></td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b><table><tr><td>Default Value:</td><td>2Eh MI_MEM_TO_MEM</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>22</td><td><b>Use Global GTT Source</b>
It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Per Process Graphics Address</td><td></td></tr><tr><td>1h</td><td>Global Graphics Address</td><td>This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.</td></tr></table></td></tr>
<tr><td></td><td>21</td><td><b>Use Global GTT Destination</b>
It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Per Process Graphics Address</td><td></td></tr><tr><td>1h</td><td>Global Graphics Address</td><td>This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.</td></tr></table></td></tr>
</table>

MI_COPY_MEM_MEM		
	20:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 3 Excludes DWord (0,1) Format: =n
1..2	63:2	**Destination Memory Address** Format: VIRTUAL_ADDR[63:2] Surface Type: MMIO Register
	1:0	**Reserved** Access: RO Format: MBZ
3..4	63:2	**Source Memory Address** Format: VIRTUAL_ADDR[63:2] Surface Type: MMIO Register
	1:0	**Reserved** Access: RO Format: MBZ

MI_COPY_MEM_MEM

<table>
<tr><th colspan="3">MI_COPY_MEM_MEM</th></tr>
<tr><td colspan="3">Source: VideoEnhancementCS
Length Bias: 2</td></tr>
<tr><td colspan="3">The MI_COPY_MEM_MEM command reads a DWord from memory and stores the value of that DWord to back to memory. The source and destination addresses are specified in the command. The command temporarily halts command execution.</td></tr>
<tr><td colspan="3"><table><tr><th>Programming Notes</th></tr><tr><td>This command should not be used within a "non-privileged" batch buffer to access global virtual space; doing so will be treated as privilege access violation. Refer to the "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to learn more about HW behavior on encountering a privilege access violation.
This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.</td></tr></table></td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b><table><tr><td>Default Value:</td><td>2Eh MI_MEM_TO_MEM</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>22</td><td><b>Use Global GTT Source</b>
It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Per Process Graphics Address</td><td></td></tr><tr><td>1h</td><td>Global Graphics Address</td><td>This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.</td></tr></table></td></tr>
<tr><td></td><td>21</td><td><b>Use Global GTT Destination</b>
It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Per Process Graphics Address</td><td></td></tr><tr><td>1h</td><td>Global Graphics Address</td><td>This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.</td></tr></table></td></tr>
</table>

MI_COPY_MEM_MEM

	20:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 3 Excludes DWord (0,1) Format: =n
1..2	63:2	**Destination Memory Address** Format: VIRTUAL_ADDR[63:2] Surface Type: MMIO Register
	1:0	**Reserved** Access: RO Format: MBZ
3..4	63:2	**Source Memory Address** Format: VIRTUAL_ADDR[63:2] Surface Type: MMIO Register
	1:0	**Reserved** Access: RO Format: MBZ

MI_DISPLAY_FLIP

MI_DISPLAY_FLIP

Source:	RenderCS, BlitterCS
Length Bias:	2

The MI_DISPLAY_FLIP command is used to request a specific display plane to switch (flip) to display a new buffer. The buffer is specified with a starting address and pitch. The tiled attribute of the buffer start address is programmed as part of the packet.

The operation this command performs is also known as a "display flip request" operation - in that the flip operation itself will occur at some point in the future. This command specifies when the flip operation is to occur: either synchronously with vertical retrace to avoid tearing artifacts (possibly on a future frame), or asynchronously (as soon as possible) to minimize rendering stalls at the cost of tearing artifacts.

Programming Notes

1. This command simply requests a display flip operation. Command execution then continues normally. There is no guarantee that the flip (even if asynchronous) will occur prior to subsequent commands being executed. (Note that completion of the MI_FLUSH command does not guarantee that outstanding flip operations have completed). The MI_WAIT_FOR_EVENT command can be used to provide this synchronization - by pausing command execution until a pending flip has actually completed. This synchronization can also be performed by use of the Display Flip Pending hardware status. See Display Flip Synchronization in the Device Programming Interface chapter of MI Functions.
2. After a display flip operation is requested, software is responsible for initiating any required synchronization with subsequent buffer clear or rendering operations. For multi-buffering (e.g., double buffering) operations, this will typically require updating SURFACE_STATE or the binding table to change the rendering (back) buffer. In addition, prior to any subsequent clear or rendering operations, software must typically ensure that the new rendering buffer is not actively being displayed. Again, the MI_WAIT_FOR_EVENT command or Display Flip Pending hardware status can be used to provide this synchronization. See Display Flip Synchronization in the Device Programming Interface chapter of MI Functions.
3. The display buffer command uses the X and Y offset for the tiled buffers from the Display Interface registers. Software is allowed to change the offset via the MMIO interface irrespective of the flip commands enqueued in the command stream. For tiled buffers, the display subsystem uses the X and Y offset in generation of the final request to memory. The offset is always updated on the next vblank for both Synchronous and Asynch Flips. It is not necessary to have a flip enqueued to update the X and Y offset.
4. The display buffer command uses the linear dword offset for the linear buffers from the Display Interface registers. Software is allowed to change the offset via the MMIO interface irrespective of the flip commands enqueued in the command stream. For linear buffers, the display subsystem uses the dword offset in generation of the final request to memory.
    - For synchronous flips the offset is updated on the next vblank. It is not necessary to have a sync flip enqueued to update the dword offset.
5. DWord 3 (Left Eye Display Buffer Base Address) must not be set with synchronous flips or asynchronous flips. It is only allowed to be sent with stereo 3D flips.

MI_DISPLAY_FLIP

Asynchronous flip completion time depends greatly on how much data has been prefetched for power savings, and can take up to 1 full frame to complete. For faster flip completion, disable FBC and render compression and allocate a small amount of data buffer for the plane.

"Command Streamer Plane Number" mapping to "Display Plane Name" are listed in display B-spec -"Plane capability and Interoperability".

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 14h MI_DISPLAY_FLIP Format: OpCode
	22	**Async Flip Indicator** Format: Enable This bit should always be set if DW2 [1:0] == '01' (async flip). This field is required due to HW limitations. This bit is used by the render pipe while DW2 is used by the display hardware.
	21:19	**Reserved** Access: RO Format: MBZ
	18:17	**Reserved**
	16:14	**Reserved** Access: RO Format: MBZ
	13:8	**Display Plane Select**

Value	Name
0h	Display Plane 1
1h	Display Plane 2
2h	Display Plane 3
3h	Reserved
4h	Display Plane 4
5h	Display Plane 5
6h	Display Plane 6
7h	Display Plane 7
8h	Display Plane 8
9h	Display Plane 9
Ah	Display Plane 10
Bh	Display Plane 11

Ch	Display Plane 12
Dh	Display Plane 13
Eh	Display Plane 14
Fh	Display Plane 15
10h	Display Plane 16
11h	Display Plane 17
12h	Display Plane 18
13h	Display Plane 19
14h	Display Plane 20
15h	Display Plane 21
16h	Display Plane 22
17h	Display Plane 23
18h	Display Plane 24
19h	Display Plane 25
1Ah	Display Plane 26
1Bh	Display Plane 27
1Ch	Display Plane 28
1Dh	Display Plane 29
1Eh	Display Plane 30
1Fh	Display Plane 31
20h	Display Plane 32
[21h, 3Fh]	Reserved

7:0 **DWord Length**

Format:	=n

Total Length - 2. Excludes DWord (0,1).

For Synchronous Flips and Asynchronous Flips, this field must be programmed to 1h for a total length of 3.

For Stereo 3D Flips, this field must be programmed to 2h for a total length of 4.

Value	Name	Exists If
1h	**[Default]**	([Flip Type]!='Stereo 3D Flip')
2h	**[Default]**	([Flip Type]=='Stereo 3D Flip')

1 31 **Stereoscopic 3D Mode**

Default Value:	0h
Format:	Enable

This bit must be set if the Extra Display Buffer Address is part of this command. This bit is used to notify the display there is an extra DW before processing the Display Flip.

<table>
<tr><th colspan="3">MI_DISPLAY_FLIP</th></tr>
<tr><td></td><td>30:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:6</td><td><b>Display Buffer Pitch</b><table><tr><td>Default Value:</td><td>0h</td></tr><tr><td>Format:</td><td>U10</td></tr></table><i>For Synchronous Flips and Stereo 3D Flips only,</i> this field specifies the pitch of the new display buffer. For Asynchronous Flips, this parameter is programmed so that all the flips in a flip chain should maintain the same pitch as programmed with the last synchronous flip or stereo 3D flip or direct through MMIO. See the Display Plane Stride register for details.</td></tr>
<tr><td></td><td>5:3</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>2:0</td><td><b>Tile Parameter</b><table><tr><td>Format:</td><td>Enable</td></tr></table>For Asynchronous Flips, this parameter cannot be changed. All the flips in a flip chain should maintain the same tile parameter as programmed with the last synchronous flip or direct through MMIO.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Linear <b>[Default]</b></td><td>For Synchronous Flips Only</td></tr><tr><td>1h</td><td>Tiled X</td><td></td></tr><tr><td>2h-3h</td><td>Reserved</td><td></td></tr><tr><td>4h</td><td>Tiled Y Legacy (Y B)</td><td></td></tr><tr><td>5h</td><td>Tile 4</td><td></td></tr></table></td></tr>
<tr><td>2</td><td>31:12</td><td><b>Display Buffer Base Address</b><table><tr><td>Format:</td><td>GraphicsAddress[31:12]</td></tr></table>This field specifies Bits 31:12 of the Graphics Address of the new display buffer. In stereo 3D mode this is the right eye base address. In non-stereo 3D mode this is the only base address. (Refer to the Display Address Start Address Register description in the Display Registers chapter).<table><tr><th>Programming Notes</th></tr><tr><td><ul><li>The Display buffer must reside completely in Main Memory.</li><li>This address is always translated via the global (rather than per-process) GTT.</li></ul></td></tr></table></td></tr>
<tr><td></td><td>11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10:7</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

MI_DISPLAY_FLIP

<table>
<tr><td></td><td>6:4</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>3</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>2</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>1:0</td><td><b>Flip Type</b>
This field specifies whether the flip operation should be performed asynchronously to vertical retrace.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>00b</td><td>Sync Flip <b>[Default]</b></td><td>The flip will occur during the vertical blanking interval - thus avoiding any tearing artifacts.</td></tr>
<tr><td>01b</td><td>Async Flip</td><td>The flip will occur "as soon as possible" - and may exhibit tearing artifacts</td></tr>
<tr><td>10b</td><td>Stereo 3D Flip</td><td>The flip will occur during the vertical blanking interval (left or right eye blank selectable through display MMIO register) - thus avoiding any tearing artifacts.</td></tr>
<tr><td>11b</td><td>Reserved</td><td></td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>
<ul>
<li>The Display Buffer Pitch and Tile parameter cannot be changed for asynchronous flips (i.e., the new buffer must have the same pitch/tile format as the previous buffer). Only display surface base address can be changed.</li>
<li>For Asynch Flips the Buffers used must be 32KB aligned.</li>
<li>Asynch flips are supported on primary or universal planes only.</li>
</ul>
</td></tr>
<tr><td>
<ul>
<li>For Stereo 3D flips, both left and right eye buffers must have the same pitch and tile format.</li>
</ul>
</td></tr>
</table></td></tr>
<tr><td>3</td><td>31:12</td><td><b>Left Eye Display Buffer Base Address</b>
<table>
<tr><td>Format:</td><td>GraphicsAddress[31:12]</td></tr>
</table>
This field specifies Bits 31:12 of the Graphics Address of the new display buffer for the stereo 3D left eye. In non-stereo 3D mode this address is not used. (Refer to the Display Address Start Address Register description in the Display Registers chapter).
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>
<ul>
<li>The Display buffer must reside completely in Main Memory.</li>
<li>This address is always translated via the global (rather than per-process) GTT.</li>
</ul>
</td></tr>
</table></td></tr>
<tr><td></td><td>11:0</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

MI_FLUSH_DW

MI_FLUSH_DW		
Source: BlitterCS Length Bias: 2		
The MI_FLUSH_DW command is used to perform an internal "flush" operation. The parser pauses on an internal flush until all drawing engines have completed any pending operations. In addition, this command can also be used to: Flush any dirty data to memory. Invalidate the TLB cache inside the hardware **Usage note: After this command is completed with a Store DWord enabled, CPU access to graphics memory will be coherent (assuming the Render Cache flush is not inhibited).**		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 26h MI_FLUSH_DW Format: OpCode
	22	**Reserved** Access: RO Format: MBZ
	21	**Store Data Index** Format: U1 This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is index into the global hardware status page when destination address type in the command is set to 1 (GGTT). The store data address is index into the per-process hardware status page when destination address type in the command is set to 0 (PPGTT).
	20:19	**Reserved** Access: RO Format: MBZ
	18	**TLB Invalidate** Format: U1 If ENABLED, all TLBs belonging to Blitter Engine will be invalidated once the flush operation is complete. This bit is only valid when the Post-Sync Operation field is a value of 1h or 3h. If GFX_MODE (0x229c) bit 13, this command will cause a config write to MMIO register space with the address 0x4f100.

MI_FLUSH_DW

17 **Reserved**

Access:	RO
Format:	MBZ

16 **Flush CCS**

Format:	Enable

If enabled, at the end of the current MI_FLUSH_DW Copy Engine Write data which may be sitting in the CCS cache will be flushed out to memory.

Programming Notes
If compression is enabled, even after Blitter is flushed some of the write data may be sitting in the CCS cache. In order to flush that data software needs to program a MI_FLUSH_DW with Flush CCS bit set after the flush.

15:14 **Post-Sync Operation**

Value	Name	Description
0h	No Write	No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc.
1h	Write Immediate Data QWord	Write the QWord containing Immediate Data Low, High DWs to the Destination Address
2h	Reserved	Reserved
3h	Write TIMESTAMP Register	Write the TIMESTAMP register to the Destination Address. The upper 28 bits of the TIMESTAMP register are tied to '0'.

Programming Notes
If executed in a PPGTT (non-secure) batch buffer, the post-sync op is always inhibited. This command does not write anything out to memory. [For all other devices]: If executed in a non-secure batch buffer, the address given is in a PPGTT address space. If in a secure ring or batch, the address given is in GGTT space.

13:10 **Reserved**

Access:	RO
Format:	MBZ

9 **Flush LLC**

Format:	Enable

If enabled, at the end of the current MI_FLUSH_DW the last level cache is cleared of all the cachelines which have been

<table>
<tr><th colspan="3">MI_FLUSH_DW</th></tr>
<tr><td rowspan="4"></td><td></td><td>determined as being part of the Frame Buffer.</td></tr>
<tr><td>8</td><td><b>Notify Enable</b><table><tr><td>Format:</td><td>U1</td></tr></table>If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in Memory Interface Registers for details.</td></tr>
<tr><td>7:6</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>5:0</td><td><b>DWord Length</b><table><tr><td>Default Value:</td><td>3h DWORD_COUNT_n</td></tr><tr><td>Format:</td><td>=n</td></tr></table>Total Length - 2. Excludes DWord (0,1).</td></tr>
<tr><td rowspan="5">1..2
This field specifies the destination address where the DWord or QWord will be stored. Note that the address can only be QWord aligned, irrespective of data size. GraphicsAddress is 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and MBZ. Some GraphicsAddress fields only specify the upper address bits. For example GraphicsAddress[47:12] would be a 4KB page address.</td><td>63:48</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>47:3</td><td><b>Destination Address</b><table><tr><td>Format:</td><td>GraphicsAddress[47:3]</td></tr></table></td></tr>
<tr><td>2</td><td><b>Destination Address Type</b>
Defines the address space of the Destination Address.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>PPGTT</td><td>Use PPGTT address space for DW write</td></tr><tr><td>1h</td><td>GGTT</td><td>Use GGTT address space for DW write</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>Ignored if "No write" is the selected in Operation.</td></tr></table></td></tr>
<tr><td>1:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td></td></tr>
<tr><td>3..4</td><td>63:0</td><td><b>Immediate Data</b>
This field specifies the DWord value to be written to the targeted location. Only valid when 15:14 in header is set to 1h.<table><tr><th>Programming Notes</th></tr><tr><td>To avoid hitting a known hardware bug, drivers cannot send a QW write when bit 5 of the address is '1'</td></tr></table></td></tr>
</table>

MI_FLUSH_DW

<table>
<tr><th colspan="3">MI_FLUSH_DW</th></tr>
<tr><td colspan="3">Source: VideoCS
Length Bias: 2</td></tr>
<tr><td colspan="3">The MI_FLUSH_DW command is used to perform an internal "flush" operation. The parser pauses on an internal flush until all drawing engines have completed any pending operations. In addition, this command can also be used to: Flush any dirty data to memory. Invalidate the TLB cache inside the hardware Usage note: After this command is completed with a Store DWord enabled, CPU access to graphics memory will be coherent (assuming the Render Cache flush is not inhibited).</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table>
<tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b>
<table>
<tr><td>Default Value:</td><td>26h MI_FLUSH_DW</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>22</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>21</td><td><b>Store Data Index</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is index into the global hardware status page when destination address type in the command is set to 1 (GGTT). The store data address is index into the per-process hardware status page when destination address type in the command is set to 0 (PPGTT).</td></tr>
<tr><td></td><td>20:19</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>18</td><td><b>TLB Invalidate</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
If ENABLED, all TLBs belonging to Video Engine will be invalidated once the flush operation is complete. This bit is only valid when the Post-Sync Operation field is a value of 1h or 3h.</td></tr>
<tr><td></td><td>17:16</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>15:14</td><td><b>Post-Sync Operation</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>No Write</td><td>No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc.</td></tr>
<tr><td>1h</td><td>Write Immediate</td><td>HW implicitly detects the Data size to be Qword or Dword to be written to memory based on the command dword length programmed . When</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">MI_FLUSH_DW</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td></td><td>Data</td><td>Dword Length indicates Qword, Writes the QWord containing Immediate Data Low, High DWs to the Destination Address . When Dword Length indicates Dword, Writes the DWord containing Immediate Data Low to the Destination Address</td></tr>
<tr><td>2h</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>3h</td><td></td><td>Write the TIMESTAMP register to the Destination Address. The upper 28 bits of the TIMESTAMP register are tied to '0'.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>If executed in a PPGTT (non-secure) batch buffer, the post-sync op is always inhibited. This command does not write anything to memory.</td></tr>
</table>
</td></tr>
<tr><td></td><td>13:10</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>9</td><td><b>Flush LLC</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
If enabled, at the end of the current MI_FLUSH_DW the last level cache is cleared of all the cachelines which have been determined as being part of the Frame Buffer.
</td></tr>
<tr><td></td><td>8</td><td><b>Notify Enable</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in Memory Interface Registers for details.
</td></tr>
<tr><td></td><td>7</td><td><b>Video Pipeline Cache invalidate</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
Enable the invalidation of the video cache at the end of this flush
</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>5:0</td><td><b>DWord Length</b>
<table>
<tr><td>Format:</td><td>=n</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>2h</td><td>DWord</td></tr>
<tr><td>3h</td><td>QWord <b>[Default]</b></td></tr>
</table>
</td></tr>
<tr><td>1..2</td><td>63:57</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
</table>

MI_FLUSH_DW

	56:48	**Reserved** Access: RO Format: MBZ
	47:3	**Address** Format: GraphicsAddress[47:3]U64 This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space. GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by the hardware. Upper bits [63:48] are ignored and MBZ. Some GraphicsAddress fields only specify the upper address bits. For example GraphicsAddress[47:12] is a 4KB page address.
	2	**Destination Address Type** Defines address space of Destination Address **Value** | **Name** | **Description** 0h | PPGTT | Use PPGTT address space for DW write 1h | GGTT | Use GGTT address space for DW write **Programming Notes** Ignored if "No write" is the selected in Operation.
	1:0	**Reserved** Access: RO Format: MBZ
3..4	63:0	**Immediate Data** This field specifies the DWord value to be written to the targeted location. DW2 is the lower DW if QW is desired. Only valid when 15:14 in header is set to 1h To avoid hitting a known hardware bug, drivers cannot send a QW write when bit 5 of the address is '1'

MI_FLUSH_DW

MI_FLUSH_DW

Source:	VideoEnhancementCS
Length Bias:	2

The MI_FLUSH_DW command is used to perform an internal "flush" operation. The parser pauses on an internal flush until all drawing engines have completed any pending operations. In addition, this command can also be used to:

- Flush any dirty data to memory.
- Invalidate the TLB cache inside the hardware

Usage note: After this command is completed with a Store DWord enabled, CPU access to graphics memory will be coherent (assuming the Render Cache flush is not inhibited).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 26h MI_FLUSH_DW Format: OpCode
	22	**Reserved**
	21	**Store Data Index** Format: U1 This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is index into the global hardware status page when destination address type in the command is set to 1 (GGTT). The store data address is index into the per-process hardware status page when destination address type in the command is set to 0 (PPGTT).
	20:19	**Reserved** Access: RO Format: MBZ
	18	**TLB Invalidate** Format: U1 If ENABLED, all TLBs belonging to Video Enhancement Engine will be invalidated once the flush operation is complete. This bit is only valid when the Post-Sync Operation field is a value of 1h or 3h. If GFX_MODE (0x229c) bit 13, this command will cause a config write to MMIO register space with the address 0x4f100.

<table>
<tr><th colspan="3">MI_FLUSH_DW</th></tr>
<tr><td></td><td>17:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:14</td><td><b>Post-Sync Operation</b><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>No Write</td><td>No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc.</td></tr><tr><td>1h</td><td>Write Immediate Data</td><td>Write the QWord containing Immediate Data Low, High DWs to the Destination Address</td></tr><tr><td>2h</td><td>Reserved</td><td>Reserved</td></tr><tr><td>3h</td><td>Write TIMESTAMP register</td><td>Write the TIMESTAMP register to the Destination Address. The upper 28 bits of the TIMESTAMP register are tied to '0'.</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>If executed in non-secure batch buffer, the address given will be in a PPGTT address space. If in a secure ring or batch, address given will be in GGTT space</td></tr></table></td></tr>
<tr><td></td><td>13:10</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>9</td><td><b>Flush LLC</b><table><tr><td>Format:</td><td>Enable</td></tr></table>If enabled, at the end of the current MI_FLUSH_DW the last level cache is cleared of all the cachelines which have been determined as being part of the Frame Buffer.</td></tr>
<tr><td></td><td>8</td><td><b>Notify Enable</b><table><tr><td>Format:</td><td>U1</td></tr></table>If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in Memory Interface Registers for details.</td></tr>
<tr><td></td><td>7:6</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>DWord Length</b><table><tr><td>Default Value:</td><td>3h Excludes DWord (0,1) = 2 for DWord, 3 for QWord</td></tr><tr><td>Format:</td><td>=n</td></tr></table></td></tr>
<tr><td>1</td><td>31:3</td><td><b>Address</b><table><tr><td>Format:</td><td>GraphicsAddress[31:3]U28</td></tr></table>This field specifies Bits 31:3 of the Address where the DWord or QWord will be stored. Note that the address can only be QWord aligned, irrespective of data size.</td></tr>
</table>

MI_FLUSH_DW

<table>
<tr><td></td><td>2</td><td><b>Destination Address Type</b>
Defines address space of Destination Address
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>PPGTT</td><td>Use PPGTT address space for DW write</td></tr>
<tr><td>1h</td><td>GGTT</td><td>Use GGTT address space for DW write</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Ignored if "No write" is the selected in Operation.</td></tr>
</table>
</td></tr>
<tr><td></td><td>1:0</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>31:16</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>15:0</td><td><b>Address High</b>
<table>
<tr><td>Format:</td><td>GraphicsAddress[47:32]U64</td></tr>
</table>
This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space
</td></tr>
<tr><td>3..4</td><td>63:0</td><td><b>Immediate Data</b>
<table>
<tr><td>This field specifies the DWord value to be written to the targeted location. DW2 is the lower DW if QW is desired. Only valid when 15:14 in header is set to 1h</td></tr>
<tr><td>To avoid hitting a known hardware bug, drivers cannot send a QW write when bit 5 of the address is '1'</td></tr>
</table>
</td></tr>
</table>

MI_FORCE_WAKEUP

MI_FORCE_WAKEUP

Source:	CommandStreamer
Length Bias:	2

This command is used to communicate Force Wakeup request to PM unit. No functionality is performed by this command when none of the mask bits are set and is equivalent to NOOP. Example for usage model: VCS Ring Buffer: MI_FORCE_WAKEUP (Force Render Awake set to '1') MI_SEMPAHORE_SIGNAL (Signal context id 0xABC to Render Command Streamer) MI_FORCE_WAKEUP (Force Render Awake set to '0') MI_BATCH_BUFFER_START STATE Commands .. .. MI_FORCE_WAKEUP (Force Render Awake set to '1') MI_LOAD_REGISTER_IMMEDIATE (Load register 0x23XX in render command streamer with data 0xFFF) MI_FORCE_WAKEUP (Force Render Awake set to '0') .. MI_BATCH_BUFFER_END

Programming Notes
This command must not be programmed in the command stream for Render Engine Command Streamer or Position Commmand Streamer or Compute Engine Command Streamer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Dh MI_FORCE_WAKEUP Format: OpCode
	22:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:16	**Mask Bits** Format: Mask[15:0] **Programming Notes** Must be set to modify corresponding Bits 15:0. (Mask bits must not be set for reserved bits).
	15:10	**Reserved** Access: RO Format: MBZ
	9	**Reserved**
	8	**Reserved**

MI_FORCE_WAKEUP

	7:5	**Reserved** Access: RO Format: MBZ
	4	**Force Media-Slice3 Awake** Format: U1 When set, Command Streamer sends message to PM to force awake the media engines in media slice 3, i.e., VCS0, VCS1, and VECS0 (next instructions require media engine awake). Command streamer waits for acknowledge from PM before parsing the next command. When reset, command streamer sends message to PM to disable force awake of media engine (next instructions do not require the media engine to be awake). Command streamer waits for acknowledge from PM before parsing the next command. **Programming Notes** Mask bit [20] has to be set for HW to look at this field when MI_FORCE_WAKEUP command is parsed. Use of this Cross-engine Force Wake is deprecated and not allowed.
	3	**Force Media-Slice2 Awake** Format: U1 When set, Command Streamer sends message to PM to force awake the media engines in media slice 2, i.e., VCS0, VCS1, and VECS0 (next instructions require media engine awake). Command streamer waits for acknowledge from PM before parsing the next command. When reset, command streamer sends message to PM to disable force awake of media engine (next instructions do not require the media engine to be awake). Command streamer waits for acknowledge from PM before parsing the next command. **Programming Notes** Mask bit [19] has to be set for HW to look at this field when MI_FORCE_WAKEUP command is parsed. Use of this Cross-engine Force Wake is deprecated and not allowed.
	2	**Force Media-Slice1 Awake** Format: U1 When set, Command Streamer sends message to PM to force awake the media engines in media slice 1, i.e., VCS0, VCS1, and VECS0 (next instructions require media engine awake). Command streamer waits for acknowledge from PM before parsing the next command. When reset, command streamer sends message to PM to disable force awake of media engine (next instructions do not require the media engine to be awake). Command streamer waits for acknowledge from PM before parsing the next command. **Programming Notes** Mask bit [18] has to be set for HW to look at this field when MI_FORCE_WAKEUP command is parsed. Use of this Cross-engine Force Wake is deprecated and not allowed.

MI_FORCE_WAKEUP

	1	**Force Render Awake** Source: BlitterCS, VideoCS, VideoEnhancementCS Format: U1 **Description** When set, Command Streamer sends message to PM to force awake render engine (next instructions require render engine or compute engine awake). Command streamer waits for acknowledge from PM before parsing the next command. When reset, command streamer sends message to PM to disable force awake of render engine (next instructions do not require the render engine or compute engine to be awake. Command streamer waits for acknowledge from PM before parsing the next command. **Programming Notes** Mask bit [17] has to be set for HW to look at this field when MI_FORCE_WAKEUP command is parsed. MI_FORCE_WAKEUP command programmed in RenderCS command buffer must not set "Force Render Awake" bit. Use of this Cross-engine Force Wake is deprecated and not allowed.
	0	**Force Media-Slice0 Awake** Format: U1 When set, Command Streamer sends message to PM to force awake the media engines in media slice 0, i.e., VCS0, VCS1, and VECS0 (next instructions require media engine awake). Command streamer waits for acknowledge from PM before parsing the next command. When reset, command streamer sends message to PM to disable force awake of media engine (next instructions do not require the media engine to be awake). Command streamer waits for acknowledge from PM before parsing the next command. **Programming Notes** Mask bit [16] has to be set for HW to look at this field when MI_FORCE_WAKEUP command is parsed. Use of this Cross-engine Force Wake is deprecated and not allowed.

MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_IMM

Source:	CommandStreamer
Length Bias:	2

The MI_LOAD_REGISTER_IMM command requests a write of up to a DWord constant supplied in the command to the specified Register Offset (i.e., offset into Memory-Mapped Register Range).
Any offset that is to a destination outside of the GT core will allow the parser to continue once the cycle is at the GT boundary and not destination. Any other address will ensure the destination is updated prior to parsing the next command

Programming Notes

Many MMIO bits require the engine to be IDLE prior to updating the value. Command streamer does not implicitly put in a pipeline flush in the cases a MMIO bit requires the engine to be IDLE. In the case there was a 3DPRIMITIVE command or GPGPU_WALKER command without any stalling PIPE_CONTROL, one must be inserted prior to a MI_LOAD_REGISTER_IMMEDIATE that is updating a bit that requires the engine to be IDLE.

When executed from a non-privileged batch buffer, MMIO writes are only allowed to the registers listed in User Mode Non-Privileged Registers for the corresponding engine, any writes targeting the register not listed in the User Mode Non-Privileged Register will convert this command to a NOOP.

The following addresses should NOT be used for LRIs:

1. 0x8800 - 0x88FF
2. >= 0xC0000

Limited LRI cycles to the Display Engine (0x40000-0xBFFFF) are allowed, but must be spaced to allow only one pending at a time. This can be done by issuing an SRM to the same address immediately after each LRI.

Programming an MMIO register is equivalent to programming a non-pipeline state to the hardware and hence an explicit stalling flush needs to be programmed prior to programming this command. However for certain MMIO registers based on their functionality doing an explicit stalling flush is exempted. Listed below are the exempted registers.

- 3DPRIM_END_OFFSET - Auto Draw End Offset
- 3DPRIM_START_VERTEX - Load Indirect Start Vertex
- 3DPRIM_VERTEX_COUNT - Load Indirect Vertex Count
- 3DPRIM_INSTANCE_COUNT - Load Indirect Instance Count
- 3DPRIM_START_INSTANCE - Load Indirect Start Instance
- 3DPRIM_BASE_VERTEX - Load Indirect Base Vertex
- 3DPRIM_XP0 - Load Indirect Extended Parameter 0
- 3DPRIM_XP1 - Load Indirect Extended Parameter 1
- 3DPRIM_XP2 - Load Indirect Extended Parameter 2

DWord	Bit	Description

<table>
<tr><th colspan="3">MI_LOAD_REGISTER_IMM</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table>
<tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b>
<table>
<tr><td>Default Value:</td><td>22h MI_LOAD_REGISTER_IMM</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>22:20</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>19</td><td><b>Add CS MMIO Start Offset</b>
This bit controls the functionality of the "Register Address" field in the command.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td></td><td>"Register Address" field in the command is treated as an offset from the executing Command Streamers MMIO start offset. Bits [22:2] of the "Register Address" are considered as dword offset to be added to the MMIO start offset of the corresponding command streamer. However, during context restore bits [11:2] of the "Register Address" are considered as dword offset to be added to the MMIO start offset of the corresponding command streamer. Refer "Register Access and User Mode Privilege Access" section to get the list of all the instances of the Command Streamers and their associated MMIO Start Offset's.
/// Command executed from Ring Buffer or Batch Buffer
Example: MI_LOAD_REGISTER_IMMEDIATE, ADD_CS_MMIO_START_OFFSET: true, Data:0xABCD, Register Address: 0x00_2000
The above command when executed on RenderCS will result in a write to MMIO offset 0x00_4000 (0x00_2000 + 0x00_2000) instead to 0x00_2000. Note that RenderCS MMIO start offset is 0x2000. For illustration table below shows the result of this command executed from few instances of the command streamers from different engines.
<table>
<tr><th>S.No</th><th>Command Streamer</th><th>Command Streamer MMIO Base</th><th>LRI Register Offset</th><th>LRI Written Address</th></tr>
<tr><td>1</td><td>RenderCS</td><td>0x2000</td><td>0x2000</td><td>0x00_4000</td></tr>
<tr><td>2</td><td>VideoCS0</td><td>0x1C_0000</td><td>0x2000</td><td>0x1C_2000</td></tr>
<tr><td>3</td><td>VideoCS1</td><td>0x1C_4000</td><td>0x2000</td><td>0x1C_6000</td></tr>
<tr><td>4</td><td>VideoEnhancement0</td><td>0x1C_8000</td><td>0x2000</td><td>0x1C_A000</td></tr>
<tr><td>5</td><td>VideoCS2</td><td>0x1D_0000</td><td>0x2000</td><td>0x1D_2000</td></tr>
<tr><td>6</td><td>VideoCS3</td><td>0x1D_4000</td><td>0x2000</td><td>0x1D_6000</td></tr>
<tr><td>7</td><td>VideoEnhancement1</td><td>0x1D_8000</td><td>0x2000</td><td>0x1D_A000</td></tr>
</table>
// Command executed from context image during context restore
Example: MI_LOAD_REGISTER_IMMEDIATE, ADD_CS_MMIO_START_OFFSET:</td></tr>
</table></td></tr>
</table>

MI_LOAD_REGISTER_IMM

true, Data:0xABCD, Register Address: 0x1C_2030
The above command when executed on RenderCS will result in a write to MMIO offset 0x2030 (0x00_2000 + 0x030) instead to 0x1C_2030. Note that RenderCS MMIO start offset is 0x2000. For illustration table below shows the result of this command executed from few instances of the command streamers from different engines during context restore.

S.No	Command Streamer	Command Streamer MMIO Base	LRI Register Offset	LRI Written Address
1	RenderCS	0x2000	0x1C_2030	0x00_2030
2	VideoCS0	0x1C_0000	0x1C_2030	0x1C_0030
3	VideoCS1	0x1C_4000	0x1C_2030	0x1C_4030
4	VideoEnhancement0	0x1C_8000	0x1C_2030	0x1C_8030
5	VideoCS2	0x1D_0000	0x1C_2030	0x1D_0030
6	VideoCS3	0x1D_4000	0x1C_2030	0x1D_4030
7	VideoEnhancement1	0x1D_8000	0x1C_2030	0x1D_8030

0	**[Default]**	"Register Address" field in the command is absolute and not an offset from the executing command streamer MMIO start offset.

18 **Reserved**

Access:	RO
Format:	MBZ

17 **MMIO Remap Enable**

This bit provides a mechanism in HW to remap the MMIO address in the MI command to the engine instance on which the command is getting executed, remapping in HW is done using engine specific remap table. Render and Compute engine share a common remapping table to facilitate remapping across engines, whereas a dedicated remap table for each of Video Decode and Video Enhancement engine class.

A MMIO remapping table per engine class is created with MMIO address belonging to multiple instances of an engine within an engine class. However Render and Compute engine share a common remapping table to facilitate remapping across engines, where as a dedicated remap table for each of Video Decode and Video Enhancement engine class.

This mode provides mechanism for SW to always use MMIO address belonging to fixed instance (instance zero) with in an engine class during command buffer creation agnostic to the instance on which it will get scheduled. This will also allow context interoperability across instances with in an engine class and extends to across engines in case of Render and Compute.

Value	Name	Description
1		MMIO remapping will be applied to the MMIO address prior to using for any other functionality of the command.
0		MMIO remapping will not be applied to the MMIO address.

MI_LOAD_REGISTER_IMM

		Programming Notes • SW must always use MMIO address belonging to Instance-0 of an engine while enabling "MMIO Remap" in MI commands. • MMIO Remapping will be done by HW prior to doing any other functionality associated with the MI command or the privilege checks. • "Add CS MMIO Start Offset" must not be enabled when "MMIO Remap" is Enabled and Vice-versa. • When remapping is not found in the remap table, HW will use the MMIO address directly without any modification.
	16:13	**Reserved** Access: RO Format: MBZ
	12	**Reserved**
	11:8	**Byte Write Disables** Range: Must specify a valid register write operation If [11:8] is '1111b', then this command will behave as a NOOP. Otherwise, the value is forwarded to the destination register.
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n
1..2	63:32	**Data DWord** Mask: Bytes Write Disables Format: U32 This field specifies the DWord value to be written to the targeted location.
	31:23	**Reserved** Access: RO Format: MBZ
	22:2	**Register Offset** Format: MmioAddress[22:2] This field specifies bits [22:2] of the offset into the Memory Mapped Register Range (i.e., this field specifies a DWord offset).
	1:0	**Reserved** Access: RO Format: MBZ

MI_LOAD_REGISTER_MEM

MI_LOAD_REGISTER_MEM

Source:	RenderCS, BlitterCS, VideoCS, VideoEnhancementCS
Length Bias:	2

The MI_LOAD_REGISTER_MEM command requests from a memory location and stores that DWord to a register.

Programming Notes
The command temporarily halts commands that will cause cycles down the 3D pipeline.
The following addresses should NOT be used for MMIO writes: • 0x8800 - 0x88FF • >= 0xC0000 Limited MMIO writes cycles to the Display Engine 0x40000-0xBFFFF) are allowed, but must be spaced to allow only one pending at a time. This can be done by issuing an SRM to the same address immediately after each MMIO write.
This command should not be used within a non-privilege batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation.
This command is not allowed to update the privilege register range when executed from a non-privilege batch buffer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 29h MI_LOAD_REGISTER_MEM Format: OpCode
	22	**Use Global GTT** Format: Boolean This bit if set when executing from a non-privileged batch buffer will be treated as privilege access violation. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer or ring buffer. This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	21	**Async Mode Enable** Format: Enable If this bit is set then the command stream will not wait for completion of this command before executing the next command. Please refer to the LOAD_INDIRECT and Predicate registers for usage of this bit.

MI_LOAD_REGISTER_MEM

20 **Reserved**

Access:	RO
Format:	MBZ

19 **Add CS MMIO Start Offset**

This bit controls the functionality of the Register Address field in the command.

Value	Name	Description
1		Register Address field in the command is treated as an offset from the executing Command Streamers MMIO start offset. Bits [22:2] of the Register Address are considered as dword offset to be added to the MMIO start offset of the corresponding command streamer. Example: MI_LOAD_REGISTER_MEM, ADD_CS_MMIO_START_OFFSET: true, Memory Address:0xABCD, Register Address: 0x1C_0030 The above command when executed on RenderCS will result in a write to MMIO offset 0x1C_2030 (0x00_2000 + 0x1C_0030) instead to 0x1C_0030. Note that RenderCS MMIO start offset is 0x2000.
0	[Default]	Register Address field in the command is absolute and not an offset from the executing command streamer MMIO start offset.

18 **Reserved**

Access:	RO
Format:	MBZ

17 **MMIO Remap Enable**

This bit provides a mechanism in HW to remap the MMIO address in the MI command to the engine instance on which the command is getting executed, remapping in HW is done using engine specific remap table. Render and Compute engine share a common remapping table to facilitate remapping across engines, whereas a dedicated remap table for each of Video Decode and Video Enhancement engine class.

A MMIO remapping table per engine class is created with MMIO address belonging to multiple instances of an engine within an engine class. However Render and Compute engine share a common remapping table to facilitate remapping across engines, where as a dedicated remap table for each of Video Decode and Video Enhancement engine class.

This mode provides mechanism for SW to always use MMIO address belonging to fixed instance (instance zero) with in an engine class during command buffer creation agnostic to the instance on which it will get scheduled. This will also allow context interoperability across instances with in an engine class and extends to across engines in case of Render and Compute.

Value	Name	Description
1		MMIO remapping will be applied to the MMIO address prior to using for any other functionality of the command.
0		MMIO remapping will not be applied to the MMIO address.

Programming Notes
• SW must always use MMIO address belonging to Instance-0 of an engine while enabling "MMIO Remap" in MI commands.

<table>
<tr><th colspan="3">MI_LOAD_REGISTER_MEM</th></tr>
<tr><td></td><td></td><td><ul><li>MMIO Remapping will be done by HW prior to doing any other functionality associated with the MI command or the privilege checks.</li><li>"Add CS MMIO Start Offset" must not be enabled when "MMIO Remap" is Enabled and Vice-versa.</li><li>When remapping is not found in the remap table, HW will use the MMIO address directly without any modification.</li></ul></td></tr>
<tr><td></td><td>16</td><td><b>Workload Partition ID Offset Enable</b>
<table>
<tr><th>Description</th></tr>
<tr><td>This bit controls the memory read address computation for fetching the value from the memory to be loaded in to the register. The final memory read address is computed by adding the Workload PartitionID times the "Address Offset" to the memory address mentioned in the command. Workload Partition ID gets programmed through WPARID register and the Address Offset gets programmed through CS_MI_ADDRESS_OFFSET register.
Example: {Final Memory Read Address[47:2], 2'b00} = (Workload Partition ID* "Address Offset") + {Memory Write Address [47:2], 2'b00}</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td></td><td>The final memory address is computed based on the Workload Partition ID.</td></tr>
<tr><td>0</td><td></td><td>There is no offset added to the memory write address.</td></tr>
</table></td></tr>
<tr><td></td><td>15:8</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Default Value:</td><td>2h Excludes DWord (0,1)</td></tr>
<tr><td>Format:</td><td>=n</td></tr>
</table></td></tr>
<tr><td>1</td><td>31:23</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>22:2</td><td><b>Register Address</b>
<table>
<tr><td>Format:</td><td>MMIOAddress[22:2]</td></tr>
</table>
This field specifies Bits 22:2 of the Register offset the DWord will be written to. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ.</td></tr>
<tr><td></td><td>1:0</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

MI_LOAD_REGISTER_MEM

2..3	63:2	**Memory Address** Format: GraphicsAddress[63:2] This field specifies the address of the memory location where the register value specified in the DWord above will read from. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[63:2] for a DWord register GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	1:0	**Reserved** Access: RO Format: MBZ

MI_LOAD_REGISTER_REG

<table>
<tr><th colspan="3">MI_LOAD_REGISTER_REG</th></tr>
<tr><td colspan="3">Source: CommandStreamer
Length Bias: 2</td></tr>
<tr><td colspan="3">The MI_LOAD_REGISTER_REG command reads from a source register location and writes that value to a destination register location.
 Any offset that is to a destination outside of the GT core will allow the parser to continue once the cycle is at the GT boundary and not destination. Any other address will ensure the destination is updated prior to parsing the next command</td></tr>
<tr><td colspan="3"><table>
<tr><th>Programming Notes</th></tr>
<tr><td>The command temporarily halts commands that will cause cycles down the 3D pipeline.</td></tr>
<tr><td>Destination register with mask implemented (Ex: Some registers have bits [31:16] as mask bits and bits[15:0] as data) will not get updated unless the value read from source register has the bits corresponding to the mask bits set. Note that any mask implemented register when read returns "0" for the bits corresponding to mask location. When the source and destination are mask implemented registers, destination register will not get updated with the source register contents.</td></tr>
<tr><td>This command is not allowed to update the privilege register range when executed from a non-privilege batch buffer.</td></tr>
</table></td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table>
<tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b><table>
<tr><td>Default Value:</td><td>2Ah MI_LOAD_REGISTER_REG</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>22:20</td><td><b>Reserved</b><table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>19</td><td><b>Add CS MMIO Start Offset Destination</b>
This bit controls the functionality of the Register Address Destination field in the command.<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td></td><td>Destination Register Address field in the command is treated as an offset from the executing Command Streamers MMIO start offset. Bits [22:2] of the Destination Register Address are considered as dword offset to be added to the MMIO start offset of the corresponding command streamer.
Example: MI_LOAD_REGISTER_REGISTER_REG, DEST_ADD_CS_MMIO_START_OFFSET: true, SRC_ADD_CS_MMIO_START_OFFSET:true, Source Register Address:0x1C_0130, Destination Register Address: 0x1C_0030
The above command when executed on RenderCS will result in a MMIO read</td></tr>
</table></td></tr>
</table>

Value	Name	Description
		from 0x1C_2130 (0x00_2000 + 0x1C_0130) and write to MMIO offset 0x1C_2030 (0x00_2000 + 0x1C_0030) instead of read from 0x1C_0130 and write to 0x1C_0030. Note that RenderCS MMIO start offset is 0x2000.
0	**[Default]**	Destination Register Address field in the command is absolute and not an offset from the executing command streamer MMIO start offset.

18 **Add CS MMIO Start Offset Source**

This bit controls the functionality of the Register Address Source field in the command.

Value	Name	Description
1		Source Register Address field in the command is treated as an offset from the executing Command Streamers MMIO start offset. Bits [22:2] of the Source Register Address are considered as dword offset to be added to the MMIO start offset of the corresponding command streamer. Example: MI_LOAD_REGISTER_REGISTER_REG, DEST_ADD_CS_MMIO_START_OFFSET: false, SRC_ADD_CS_MMIO_START_OFFSET:true, Source Register Address:0x1C_0130, Destination Register Address: 0x1C_0030 The above command when executed on RenderCS will result in a MMIO read from 0x1C _2130 instead of read from 0x1C_0130 and write to MMIO offset 0x1C_0030. Note that RenderCS MMIO start offset is 0x2000.
0	**[Default]**	Register Address field in the command is absolute and not an offset from the executing command streamer MMIO start offset.

17 **MMIO Remap Enable Destination**

This bit provides a mechanism in HW to remap the " Destination Register" MMIO address in the MI command to the engine instance on which the command is getting executed, remapping in HW is done using engine specific remap table. Render and Compute engine share a common remapping table to facilitate remapping across engines, whereas a dedicated remap table for each of Video Decode and Video Enhancement engine class.

A MMIO remapping table per engine class is created with MMIO address belonging to multiple instances of an engine within an engine class. However Render and Compute engine share a common remapping table to facilitate remapping across engines, where as a dedicated remap table for each of Video Decode and Video Enhancement engine class.

This mode provides mechanism for SW to always use MMIO address belonging to fixed instance (instance zero) with in an engine class during command buffer creation agnostic to the instance on which it will get scheduled. This willalso allow context interoperability across instances with in an engine class and extends to across engines in case of Render and Compute.

Value	Name	Description
1		MMIO remapping will be applied to the MMIO address prior to using for any other functionality of the command.
0		MMIO remapping will not be applied to the MMIO address.

Programming Notes
• SW must always use MMIO address belonging to Instance-0 of an engine while enabling "MMIO Remap" in MI commands.

MI_LOAD_REGISTER_REG

- MMIO Remapping will be done by HW prior to doing any other functionality associated with the MI command or the privilege checks.
- "Add CS MMIO Start Offset" must not be enabled when "MMIO Remap" is Enabled and Vice-versa.
- When remapping is not found in the remap table, HW will use the MMIO address directly without any modification.

16 **MMIO Remap Enable Source**

This bit provides a mechanism in HW to remap the "Source Register" MMIO address in the MI command to the engine instance on which the command is getting executed, remapping in HW is done using engine specific remap table. Render and Compute engine share a common remapping table to facilitate remapping across engines, whereas a dedicated remap table for each of Video Decode and Video Enhancement engine class.

A MMIO remapping table per engine class is created with MMIO address belonging to multiple instances of an engine within an engine class. However Render and Compute engine share a common remapping table to facilitate remapping across engines, where as a dedicated remap table for each of Video Decode and Video Enhancement engine class.

This mode provides mechanism for SW to always use MMIO address belonging to fixed instance (instance zero) with in an engine class during command buffer creation agnostic to the instance on which it will get scheduled. This will also allow context interoperability across instances within an engine class and extends to across engines in case of Render and Compute.

Value	Name	Description
1		MMIO remapping will be applied to the MMIO address prior to using for any other functionality of the command.
0		MMIO remapping will not be applied to the MMIO address.

Programming Notes

- SW must always use MMIO address belonging to Instance-0 of an engine while enabling "MMIO Remap" in MI commands.
- MMIO Remapping will be done by HW prior to doing any other functionality associated with the MI command or the privilege checks.
- "Add CS MMIO Start Offset" must not be enabled when "MMIO Remap" is Enabled and Vice-versa.
- When remapping is not found in the remap table, HW will use the MMIO address directly without any modification.

15:8 **Reserved**

Access:	RO
Format:	MBZ

7:0 **DWord Length**

Default Value:	1h Excludes DWord (0,1)
Format:	=n

MI_LOAD_REGISTER_REG		
1	31:23	**Reserved** Access: RO Format: MBZ
	22:2	**Source Register Address** Format: MMIOAddress[22:2] This field specifies Bits 22:2 of the Register offset the DWord will be written to. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ.
	1:0	**Reserved** Access: RO Format: MBZ
2	31:23	**Reserved** Access: RO Format: MBZ
	22:2	**Destination Register Address** Format: MMIOAddress[22:2] This field specifies Bits 22:2 of the Register offset the DWord will be written to. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ.
	1:0	**Reserved** Access: RO Format: MBZ

MI_LOAD_SCAN_LINES_EXCL

<table>
<tr><th colspan="3">MI_LOAD_SCAN_LINES_EXCL</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">The MI_LOAD_SCAN_LINES_EXCL command is used to initialize the Scan Line Window registers for a specific Display Pipe. If the display refresh is inside this window the Display Engine signals the command parser to release the WAIT_FOR_EVENT command (i.e., the parser will wait while outside of the window). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display pipe.
Note: The two scan-line numbers are inclusive. If programmed to the same values, that single line defines the region in question.
Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical MI_LOAD_SCAN_LINES_EXCL/INCL command.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table>
<tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b>
<table>
<tr><td>Default Value:</td><td>13h MI_LOAD_SCAN_LINES_EXCL</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>22</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>21:19</td><td><b>Display Pipe Select</b>
<table>
<tr><td>Format:</td><td>U3</td></tr>
</table>
This field selects which Display Engine (pipe) this command is targeting.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0h</td><td>Display Pipe A</td></tr>
<tr><td>1h</td><td>Display Pipe B</td></tr>
<tr><td>2h, 3h</td><td>Reserved</td></tr>
<tr><td>4h</td><td>Display Pipe C</td></tr>
<tr><td>5h</td><td>Display Pipe D</td></tr>
<tr><td>6h, 7h</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td></td><td>18:17</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>16:6</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

MI_LOAD_SCAN_LINES_EXCL

	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n
1	31:16	**Start Scan Line Number** Format: U16 This field specifies the starting scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]
	15:0	**End Scan Line Number** Format: U16 This field specifies the ending scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]

MI_LOAD_SCAN_LINES_EXCL

<table>
<tr><th colspan="3">MI_LOAD_SCAN_LINES_EXCL</th></tr>
<tr><td colspan="3">Source: RenderCS
Length Bias: 2</td></tr>
<tr><td colspan="3">The MI_LOAD_SCAN_LINES_EXCL command is used to initialize the Scan Line Window registers for a specific Display Pipe. If the display refresh is <b>inside</b> this window the Display Engine signals the command parser to release the WAIT_FOR_EVENT command (i.e., the parser will wait while outside). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display pipe. <b>Note:</b> The two scan-line numbers are inclusive. If programmed to the same values, that single line defines the region in question. Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical MI_LOAD_SCAN_LINES_EXCL/INCL command.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table>
<tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b>
<table>
<tr><td>Default Value:</td><td>13h MI_LOAD_SCAN_LINES_EXCL</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>22</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>21:19</td><td><b>Display Pipe Select</b>
<table>
<tr><td>Format:</td><td>U3</td></tr>
</table>
This field selects which Display Engine (pipe) this command is targeting.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0h</td><td>Display Pipe A</td></tr>
<tr><td>1h</td><td>Display Pipe B</td></tr>
<tr><td>2h</td><td>Reserved</td></tr>
<tr><td>3h</td><td>Reserved</td></tr>
<tr><td>4h</td><td>Display Pipe C</td></tr>
<tr><td>5h</td><td>Display Pipe D</td></tr>
</table></td></tr>
<tr><td></td><td>18:17</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>16:6</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

MI_LOAD_SCAN_LINES_EXCL

	5:0	**DWord Length**	
		Default Value:	0h
		Format:	=n
1	31:29	**Reserved**	
		Access:	RO
		Format:	MBZ
	28:16	**Start Scan Line Number**	
		Format:	U13
		Range: [0, Display Buffer height in lines-1]	
		This field specifies the starting scan line number of the Scan Line Window.	
	15:13	**Reserved**	
		Access:	RO
		Format:	MBZ
	12:0	**End Scan Line Number**	
		Format:	U13
		This field specifies the ending scan line number of the Scan Line Window.	
		Range: [0, Display Buffer height in lines-1]	

MI_LOAD_SCAN_LINES_INCL

MI_LOAD_SCAN_LINES_INCL

Source:	BlitterCS
Length Bias:	2

The MI_LOAD_SCAN_LINES_INCL command is used to initialize the Scan Line Window registers for a specific Display Engine. If the display refresh is outside this window the Display Engine signals the command parser to release the WAIT_FOR_EVENT command (i.e., the parser will wait while inside of the window). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display.
Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b><table><tr><td>Default Value:</td><td>12h MI_LOAD_SCAN_LINES_INCL</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>22</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>21:19</td><td><b>Display Pipe Select</b><table><tr><td>Format:</td><td>U3</td></tr></table>This field selects which Display Engine (pipe) this command is targeting.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0h</td><td>Display Pipe A</td></tr><tr><td>1h</td><td>Display Pipe B</td></tr><tr><td>2h, 3h</td><td>Reserved</td></tr><tr><td>4h</td><td>Display Pipe C</td></tr><tr><td>5h</td><td>Display Pipe D</td></tr><tr><td>6h, 7h</td><td>Reserved</td></tr></table></td></tr>
<tr><td></td><td>18:17</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>16:6</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>DWord Length</b><table><tr><td>Default Value:</td><td>0h Excludes DWord (0,1)</td></tr><tr><td>Format:</td><td>=n</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">MI_LOAD_SCAN_LINES_INCL</th></tr>
<tr><td>1</td><td>31:16</td><td><b>Start Scan Line Number</b><table><tr><td>Format:</td><td>U16</td></tr></table>This field specifies the starting scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]</td></tr>
<tr><td></td><td>15:0</td><td><b>End Scan Line Number</b><table><tr><td>Format:</td><td>U16</td></tr></table>This field specifies the ending scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]</td></tr>
</table>

MI_LOAD_SCAN_LINES_INCL

<table>
<tr><th colspan="3">MI_LOAD_SCAN_LINES_INCL</th></tr>
<tr><td colspan="3">Source: RenderCS
Length Bias: 2</td></tr>
<tr><td colspan="3">The MI_LOAD_SCAN_LINES_INCL command is used to initialize the Scan Line Window registers for a specific Display Engine. If the display refresh is <b>outside</b> this window the Display Engine signals the command parser to release the WAIT_FOR_EVENT command (i.e., the parser will wait while inside the window). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display.
Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical MI_LOAD_SCAN_LINES_EXCL/INCL command.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b><table><tr><td>Default Value:</td><td>12h MI_LOAD_SCAN_LINES_INCL</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>22</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>21:19</td><td><b>Display Pipe Select</b><table><tr><td>Format:</td><td>U3</td></tr></table>This field selects which Display Engine (pipe) this command is targeting.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0h</td><td>Display Pipe A</td></tr><tr><td>1h</td><td>Display Pipe B</td></tr><tr><td>2h</td><td>Reserved</td></tr><tr><td>3h</td><td>Reserved</td></tr><tr><td>4h</td><td>Display Pipe C</td></tr><tr><td>5h</td><td>Display Pipe D</td></tr></table></td></tr>
<tr><td></td><td>18:17</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>16:6</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

MI_LOAD_SCAN_LINES_INCL			
	5:0	**DWord Length**	
		Default Value:	0h
		Format:	=n
1	31:29	**Reserved**	
		Access:	RO
		Format:	MBZ
	28:16	**Start Scan Line Number**	
		Format:	U13
		Range: [0, Display Buffer height in lines-1]	
		This field specifies the starting scan line number of the Scan Line window.	
	15:13	**Reserved**	
		Access:	RO
		Format:	MBZ
	12:0	**End Scan Line Number**	
		Format:	U13
		Range: [0, Display Buffer height in lines-1]	
		This field specifies the ending scan line number of the Scan Line Window.	

MI_MATH

MI_MATH

Source:	CommandStreamer
Length Bias:	2

The MI_MATH command allows SW to send instruction to ALU in Any Command Streamer.
MI_MATH command is the means by which ALU can be accessed. ALU instructions form the data payload of MI_MATH command, ALU instruction is dword in size. MI_MATH Dword Length should be programmed based on the number of ALU instruction packed, max number is limited by the max Dword Length supported. When MI_MATH command is parsed by command streamer it outputs the payload dwords (ALU instructions) to the ALU. ALU takes single clock to process any given instruction. Refer to B-spec "Command Streamer (CS) ALU Programming" section in Command Streamer Programming.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ah MI_MATH Format: OpCode
	22:16	**Reserved** Access: RO Format: MBZ
	15	**Predication Enable** This bit is used to enable predication of this command. If this bit is set and Bit 0 of the MI_SET_PREDICATE_RESULT register is set, this command is ignored. Otherwise the command is performed normally.
	14:8	**Memory Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE
	7:0	**DWord Length** Format: =n Value: [0-255] — Name:
1..n	31:0	**ALU INSTRUCTION** Format: U32

MI_NOOP

<table>
<tr><th colspan="3">MI_NOOP</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 1</td></tr>
<tr><td colspan="3">The MI_NOOP command basically performs a "no operation" in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform - a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table>
<tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b>
<table>
<tr><td>Default Value:</td><td>0h MI_NOOP</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>22</td><td><b>Identification Number Register Write Enable</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified - making this command an effective "no operation" function.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable</td><td>Do not write the NOP_ID register.</td></tr>
<tr><td>1h</td><td>Enable</td><td>Write the NOP_ID register.</td></tr>
</table></td></tr>
<tr><td></td><td>21:0</td><td><b>Identification Number</b>
<table>
<tr><td>Format:</td><td>U22</td></tr>
</table>
This field contains a 22-bit number which can be written to the MI NOPID register.</td></tr>
</table>

MI_NOOP

<table>
<tr><th colspan="3">MI_NOOP</th></tr>
<tr><td colspan="3">Source: RenderCS
Length Bias: 1</td></tr>
<tr><td colspan="3">The MI_NOOP command basically performs a "no operation" in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform - a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).</td></tr>
<tr><td colspan="3"><table><tr><th>Performance</th></tr><tr><td>The MI_NOOP process time is reduced to 1 clock. An example use of the improved NOOP throughput is for some multi-pass media applications where some unwanted media object commands are replaced by MI_NOOP commands without repacking the commands in a batch buffer.</td></tr></table></td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b><table><tr><td>Default Value:</td><td>0h MI_NOOP</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>22</td><td><b>Identification Number Register Write Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified, making this command an effective "no operation" function.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Disable</td><td>Do not write the NOP_ID register.</td></tr><tr><td>1h</td><td>Enable</td><td>Write the NOP_ID register.</td></tr></table></td></tr>
<tr><td></td><td>21:0</td><td><b>Identification Number</b><table><tr><td>Format:</td><td>U22</td></tr></table>This field contains a 22-bit number which can be written to the MI NOPID register.</td></tr>
</table>

MI_NOOP

MI_NOOP

Source:	VideoCS
Length Bias:	1

The MI_NOOP command basically performs a "no operation" in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform - a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 00h MI_NOOP Format: OpCode
	22	**Identification Number Register Write Enable** Format: Enable This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified - making this command an effective "no operation" function. **Value** / **Name** 1 / Write the NOP_ID register.
	21:0	**Identification Number** Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.

MI_NOOP

<table>
<tr><th colspan="3">MI_NOOP</th></tr>
<tr><td colspan="3">Source: VideoEnhancementCS
Length Bias: 1</td></tr>
<tr><td colspan="3">The MI_NOOP command basically performs a "no operation" in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform - a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table>
<tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b>
<table>
<tr><td>Default Value:</td><td>00h MI_NOOP</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>22</td><td><b>Identification Number Register Write Enable</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified - making this command an effective "no operation" function.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td></td><td>Write th NOP_ID Register</td></tr>
<tr><td>0</td><td></td><td>Do not write the NOP_ID register</td></tr>
</table></td></tr>
<tr><td></td><td>21:0</td><td><b>Identification Number</b>
<table>
<tr><td>Format:</td><td>U22</td></tr>
</table>
This field contains a 22-bit number which can be written to the MI NOPID register.</td></tr>
</table>

MI_PREDICATE

<table>
<tr><th colspan="3">MI_PREDICATE</th></tr>
<tr><td colspan="3">Source: RenderCS
Length Bias: 1</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td colspan="3"><table><tr><th>Programming Notes</th></tr><tr><td>This command is supported by ComputeCS</td></tr></table></td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b><table><tr><td>Default Value:</td><td>0Ch MI_PREDICATE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>22:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:6</td><td><b>Load Operation</b>
This field controls if/how the Predicate state bit is modified.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>KEEP</td><td>The Predicate state bit is unmodified.</td></tr><tr><td>1h</td><td>Reserved</td><td></td></tr><tr><td>2h</td><td>LOAD</td><td>The Predicate state bit is loaded with the combine operation result.</td></tr><tr><td>3h</td><td>LOADINV</td><td>The Predicate state bit is loaded with the inverted combine operation result.</td></tr></table></td></tr>
<tr><td></td><td>5</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>4:3</td><td><b>Combine Operation</b>
This field controls if/how the result of the compare operation is combined with the current Predicate state bit.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>SET</td><td>The combine operation output the compare result unmodified.</td></tr><tr><td>1h</td><td>AND</td><td>The combine operation outputs the AND of the compare result and the current Predicate state bit.</td></tr><tr><td>2h</td><td>OR</td><td>The combine operation outputs the OR of the compare result and the current Predicate state bit.</td></tr></table></td></tr>
</table>

MI_PREDICATE

<table>
<tr><td></td><td></td><td><table><tr><td>3h</td><td>XOR</td><td>The combine operation outputs the XOR of the compare result and the current Predicate state bit.</td></tr></table></td></tr>
<tr><td></td><td>2</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>1:0</td><td><b>Compare Operation</b>
This field controls how Data DWord 0 and Data DWord 1 fields are used to generate a compare operation result and possibly modify the PredicateData register.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>TRUE</td><td>The compare operation outputs TRUE. The PredicateData register is unmodified.</td></tr><tr><td>1h</td><td>FALSE</td><td>The compare operation outputs FALSE. The PredicateData register is unmodified.</td></tr><tr><td>2h</td><td>SRCS_EQUAL</td><td>(MItemp0 - MItemp1) is computed and loaded into the PredicateData register. The compare operation outputs (MItemp0 == MItemp1).</td></tr><tr><td>3h</td><td>DELTAS_EQUAL</td><td>(MItemp0 - MItemp1) is computed and compared to the PredicateData register. If the values are equal, the compare operation outputs TRUE, otherwise it outputs FALSE. The PredicateData register is unmodified.</td></tr></table></td></tr>
</table>

MI_PRT_BATCH_BUFFER_START

MI_PRT_BATCH_BUFFER_START

Source:	BSpec
Length Bias:	2

Persistent batch buffer provides a mechanism to jump and execute a batch of commands from a buffer in graphics memory (PPGTT or GGTT) from within a command sequence, like a batch buffer. Command sequence in a persistent batch buffer is ended through MI_BATCH_BUFFER_END command. Primary differentiating feature it supports is when enabled through “Persistence Enable” in the command, the persistent batch buffer details are saved as part of the context state when executed and the persistent batch buffer gets executed on subsequent context restore of the corresponding context before resuming the regular command buffer execution (ring buffer or batch buffer). Persistent batch buffer can be programmed form within a Ring Buffer or a Batch Buffer. Persistent batch buffer can be invoked from the command sequence several times with different start address, while the hardware will only context save the details of the most recently executed persistent batch buffer.

Programming Notes

- Persistent Batch Buffer may be programmed only from Ring Buffer or First/Second/Third Level Batch Buffers.
- Once "Persistence Enable" is enabled for a persistent batch buffer, its SW responsibility to keep the pages backing the persistent batch buffer are made available in memory during the contexts life span.
- Preemption will not be supported from within Persistent batch buffers.
- All preemptable commands will be forced to NOOP by HW.
- MI_BATCH_BUFFER_START or MI_PRT_BATCH_BUFFER_START commands are not supported from within persistent batch buffer and will be NOOP'd.
- Persistent Batch Buffer execution doesn't happen on lite restores.

- A batch buffer initiated with this command must end with a MI_BATCH_BUFFER_END command.
- It is essential that the address location beyond the current page having MI_BATCH_BUFFER_END is populated. HW performs over-fetch of the command addresses and any over-fetch requires a valid TLB entry. A single extra page beyond the batch buffer is sufficient.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 39h MI_PRT_BATCH_BUFFER_START Format: OpCode
	22:11	**Reserved** Access: RO Format: MBZ

MI_PRT_BATCH_BUFFER_START

10 **Persistence Enable**

This field controls the enabling and disabling of Persistence batch buffer on context restore.

Value	Name	Description
0h		Persistence Batch Buffer on subsequent context restores is Disabled.
1h		Persistence Batch Buffer on subsequent context restores is Enabled.

9 **Reserved**

Access:	RO
Format:	MBZ

8 **Address Space Indicator**

This fields indicates the address space (PPGT or GGTT)of the memory in which the persistent batch buffer is located.
Batch buffers accessed via PPGTT are considered as non-privileged. Certain operations (e.g., MI_STORE_DATA_IMM commands to GGTT memory) are prohibited within non-privileged buffers. More details mentioned in User Mode Privileged command section.

Value	Name	Description
0h	GGTT	This batch buffer is located in GGTT memory and is privileged.
1h	PPGTT	This batch buffer is located in PPGTT memory and is Non-Privileged.

Programming Notes

Command when executed from a batch buffer can set its "Privileged" level to its parent batch buffer or lower. This is HW enforced.
Example:

- MI_PRT_BATCH_BUFFER_START programmed from a Ring Buffer can have "Address Space Indicator" set to GGTT or PPGTT.
- MI_PRT_BATCH_BUFFER_START programmed from a batch buffer in GGTT address space can have "Address Space Indicator" set to GGTT or PPGTT.
- MI_PRT_BATCH_BUFFER_START programmed from a batch buffer in PPGGTT address space must have "Address Space Indicator" set to PPGTT.

7:0 **DWord Length**

Default Value:	1h Excludes DWord (0,1)
Format:	=n

Total - Bias. Excludes DWord (0,1).

MI_PRT_BATCH_BUFFER_START

1..2 GraphicsAddress is a 64-bit value [63:0], but only a portion of it is used by hardware. The upper reserved bits are ignored and MBZ.	63:2	**Batch Buffer Start Address** Format: VIRTUAL_ADDR[63:2]
	1:0	**Reserved** Access: RO Format: MBZ

MI_REPORT_HEAD

MI_REPORT_HEAD

Source:	BlitterCS
Length Bias:	1

The MI_REPORT_HEAD command causes the Head Pointer value of the active ring buffer to be written to a cacheable (snooped) system memory location.
When the **Execlist Enable** bit is reset:
The location written is relative to the address programmed in the Hardware Status Page Address Register.

Programming Notes

This command must not be executed from a Batch Buffer (Refer to the description of the HWS_PGA register).
When the **Execlist Disable** is clear, the head pointer will be reported to the PP HW Status Page.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	**Reserved** Access: RO Format: MBZ

MI_REPORT_HEAD

MI_REPORT_HEAD

Source:	RenderCS
Length Bias:	1

The MI_REPORT_HEAD command causes the Head Pointer value of the active ring buffer to be written to a cacheable (snooped) system memory location. When Execlist Enable is set, the head pointer will be reported to the PP HW Status Page. The location written is relative to the address programmed in the Hardware Status Page Address Register.

Programming Notes
This command must not be executed from a Batch Buffer. (Refer to the description of the HWS_PGA register.)

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	**Reserved** Access: RO Format: MBZ

MI_REPORT_HEAD

MI_REPORT_HEAD

Source:	VideoCS
Length Bias:	1

The MI_REPORT_HEAD command causes the Head Pointer value of the ring buffer to be written to a cacheable (snooped) system memory location. When the Per-Process Virtual Address Space and Execlist Enable bit is reset: The location written is relative to the address programmed in the Hardware Status Page Address Register. When the Execlist Enable is set, the head pointer will be reported to the PP HW Status Page.

Programming Notes
This command must not be executed from a Batch Buffer (Refer to the description of the HWS_PGA register).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	**Reserved** Access: RO Format: MBZ

MI_REPORT_HEAD

MI_REPORT_HEAD

Source:	VideoEnhancementCS
Length Bias:	1

The MI_REPORT_HEAD command causes the Head Pointer value of the ring buffer to be written to a cacheable (snooped) system memory location.
When the **Per-Process Virtual Address Space and Execlist Enable bit** is reset: The location written is relative to the address programmed in the Hardware Status Page Address Register. When the **Execlist Enable** is set, the head pointer will be reported to the PP HW Status Page.

Programming Notes
This command must not be executed from a Batch Buffer (Refer to the description of the HWS_PGA register).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	**Reserved** Access: RO Format: MBZ

MI_REPORT_PERF_COUNT

MI_REPORT_PERF_COUNT

Source:	RenderCS, ComputeCS
Length Bias:	2

The MI_REPORT_PERF_COUNT command causes the GFX hardware to write out a snapshot of performance counters to the address specified in this command along with constant ID field supplied and the time-stamp counter. This write is required to be treated as a cacheable write irrespective of GTT entry memory type. This command is specific to the render engine.

Programming Notes
This command can be inserted after events of interest (frequently before and after a 3DPRIMITIVE command). SW is entirely responsible for managing the ID field and addresses used by such a series of commands.
GTT_SELECT must not be set to 1 (i.e. GGTT) when MI_REPORT_PERF_COUNT command is programmed in a non-privileged batch buffer. Refer to the "User Mode Privileged commands" Table in MI_BATCH_BUFFER_START command section for more details. All batch buffers in PPGTT are considered as Non-privileged.
MI_REPORT_PERF_COUNT is being extended to ComputeCS also with the OAC feature.
MI_REPORT_PERF_COUNT is only supported by one of the ComputeCS that is selected for performance monitoring through “CCS Select for Perf Mon” in OAG_OACONTROL register.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 28h MI_REPORT_PERF_COUNT Format: OpCode
	22:6	**Reserved** Access: RO Format: MBZ
	5:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1..2	63:6	**Memory Address** Format: GraphicsAddress[63:6] This field specifies 64B aligned GFX MEM address where the chap counter values are reported. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47] **Programming Notes** This field is ignored if "Report to OABUFFER" bit is set.

MI_REPORT_PERF_COUNT

	5	**Reserved** Access: RO Format: MBZ
	4	**Core Mode Enable** Format: U1 This bit is set then the address will be offset by the Core ID:If Core ID 0, then there is no offset If Core ID 1, then the Memory is offset by the size of the data(64b).
	3:1	**Reserved** Access: RO Format: MBZ
	0	**Use Global GTT** Format: Boolean This field when set (i.e. bit = 1) selects the GGTT for address translation. When this bit is 0 (default value), HW should use PGTT for address translation.
3	31:0	**Report ID** Format: U32 This field specifies the ID provided by SW for a given report command. It can be tracked to use different flavors of these reports based on where in command-stream they are inserted. This field is reported only when Counter Select Field is 0. **Programming Notes** If a privilege access violation occurs, the REPORT ID field in the report generated by the next legitimate MI_REPORT_PERF_COUNT will be corrupted.

MI_RS_STORE_DATA_IMM

MI_RS_STORE_DATA_IMM

Source:	RenderCS
Length Bias:	2

The MI_RS_STORE_DATA_IMM command requests a write of the DWord constant supplied in the packet to the specified Memory Address.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Bh MI_RS_STORE_DATA_IMM Format: OpCode MI_RS_STORE_DATA_IMM
	22	**Reserved** Access: RO Format: MBZ
	21	**Reserved**
	20:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n
1..2	63:2	**Destination Address** Format: GraphicsAddress[63:2] This field specifies Bits 47:2 of the Address where the DWord will be stored. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47]. When render engine is PPGTT enabled this Address is translated using PPGTT, else GGTT is used for translation.
	1	**Reserved** Access: RO Format: MBZ
	0	**Core Mode Enable** If this bit is set then the address will be offset by the Core ID: If Core ID 0, then there is no offset If Core ID 1, then the Memory is offset by the size of the data.

MI_RS_STORE_DATA_IMM

3	31:0	**Data DWord 0** Format: U32 This field specifies the DWord value to be written to the targeted location.

MI_SEMAPHORE_SIGNAL

MI_SEMAPHORE_SIGNAL

Source:	CommandStreamer
Length Bias:	2

Description

An engine on executing this command generates a signal (interrupt) to the GUC (scheduler or FW) by reporting the Producer Token Number programmed in SEMAPHORE_TOKEN register. Each engine implements its own SEMAPHORE_TOKEN register. SEMAPHORE_TOKEN register is privileged and context save/restored. Scheduler can take appropriate action on decoding the reported Producer Token Number. Typically MI_ATOMIC (non-posted) command will be used to update the memory semaphore before signaling the consumer context.

Each engine implements SEMAPHORE_SIGNAL_PORT register for receiving semaphore signal from the scheduler (SW or FW). A write to the SEMAPHORE_SIGNAL_PORT with data as 0xFFFF_FFFF is decoded as semaphore signal received by the corresponding engine. An engine waiting on un-successful MI_SEMAPHORE_WAIT (signal mode) command will reacquire the semaphore data from memory and re-evaluate the semaphore comparison on receiving the semaphore signal. SEMAPHORE_SIGNAL_PORT register is privileged. Writing to the SEMAPHORE_SIGNAL_PORT of an idle engine (no context) does not trigger any action in HW and is of no use.

SEMAPHORE_TOKEN, MI_SEMAPHORE_SIGNAL, SEMAPHORE_SIGNAL_PORT and MI_SEMAPHORE_WAIT together can be used to create semaphores between producer context and consumer context. MI_SEMPAHORE_SIGNAL command from a producer context can be used to signal a consumer context waiting on MI_SEMAPHORE_WAIT (signal mode) command through scheduler (SW or FW).

- Typically MI_ATOMIC (non-posted) command will be used to update the memory semaphore by the producer context before signaling the consumer context.
- Scheduler on receiving the signal will process the Producer Token Number and if required will signal the consumer context running on an engine by writing 0xFFFF_FFFF to the corresponding engines SEMAPHORE_SIGNAL_PORT.
- A consumer context will wait on MI_SEMAPHORE_WAIT (signal mode) command until the semaphore comparison is successful. An engine waiting on un-successful MI_SEMAPHORE_WAIT (signal mode) command will reacquire the semaphore data from memory and re-evaluate the semaphore comparison on receiving the semaphore signal. MI_SEMAPHORE_WAIT command has Wait Token Number as inline data programmed by the SW. Context switched out an un-successful MI_SEMAPHORE_WAIT command will report Wait Token Number as Wait Detail field in the CSB structure.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Bh MI_SEMAPHORE_SIGNAL Format: OpCode

<table>
<tr><th colspan="3">MI_SEMAPHORE_SIGNAL</th></tr>
<tr><td></td><td>22</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>21</td><td><b>Post-Sync Operation</b><table><tr><td>Source:</td><td>RenderCS</td></tr></table>
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>No Post Sync Operation</td><td>Command is executed as usual.</td></tr>
<tr><td>1h</td><td>Post Sync Operation</td><td>MI_SEMAPHORE_SIGNAL command is executed as a pipelined PIPE_CONTROL flush command with Semaphore Signal as post sync operation. Flush completion only guarantees the workload prior to this command is pushed till Windower unit and completion of any outstanding flushes issued prior to this command.</td></tr></table>
<table><tr><th>Programming Notes</th></tr>
<tr><td>Any desired pipeline flush operation can be achieved by programming PIPE_CONTROL command prior to this command.
When this bit is set Command Streamer sends a flush down the pipe and the atomic operation is saved as post sync operation. Command streamer goes on executing the following commands. Atomic operation saved as post sync operation is executed at some point later on completion of corresponding flush issued.
When this bit is set atomic semaphore signal operation will be out of order with rest of the MI commands programmed in the ring buffer or batch buffer, it will be in order with respect to the post sync operations resulting due to PIPE_CONTROL command.</td></tr>
<tr><td>This bit must not be set when executed by ComputeCS.</td></tr></table></td></tr>
<tr><td></td><td>20:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b><table><tr><td>Default Value:</td><td>0h Excludes DWord (0,1)</td></tr><tr><td>Format:</td><td>=n</td></tr></table>Total Length - 2</td></tr>
<tr><td>1</td><td>31:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

MI_SEMAPHORE_WAIT

MI_SEMAPHORE_WAIT

Source:	CommandStreamer
Length Bias:	2

Description

This command supports memory based Semaphore WAIT. Memory based semaphores will be used for synchronization between the Producer and the Consumer contexts. Producer and Consumer Contexts could be running on different engines or on the same engine inside GT. Producer Context implements a Signal and Consumer context implements a Wait.
Command Streamer on parsing this command fetches data from the Semaphore Address mentioned in this command and compares it with the inline Semaphore Data Dword.

- If comparison passes, the command streamer moves to the next command.
- If comparison fails Command streamer switches out the context. Context switch can be inhibited by setting "Inhibit Synchronous Context Switch" in CTXT_SR_CTL register.
- If "Inhibit Synchronous context Switch" is enabled and comparison fails, Command Streamer evaluates the Compare Operation based on the Wait Mode until the compare operation is true or Wait is canceled by SW.
- CS generates semaphore wait interrupt to the scheduler when MI_SEMAPHORE_WAIT command is un-successful and when "Inhibit Synchronous Context Switch" is set. Scheduler can use this interrupt to preempt the context waiting on semaphore wait.

MI_SEMAPHORE_WAIT command also supports register based Semaphore WAIT. Command Streamer on parsing this command fetches data from the MMIO offset mentioned in this command and compares it with the inline Semaphore Data Dword. This functionality is supported when Register Poll bit is set in the command header. In register poll mode of operation Wait Mode supported is always Poll mode and no Signal mode is supported.

- If comparison passes, the command streamer moves to the next command.
- Unlike in Memory based semaphore, there is no context switch on an un-successful semaphore wait in Register Poll mode, however preemption is supported on unsuccessful semaphore wait in Register Poll mode. Semaphore wait interrupt is not generated by default on wait un-successful in Register Poll mode.
- Also unlike in Memory based semaphore, generation of an interrupt for a semaphore wait in "Register Poll" mode is not dependent on the value of bit "Inhibit Synchronous Context Switch" in register "CTXT_SR_CTL"
- Register Poll mode of Semaphore Wait command operation is non-privileged and will be supported from PPGTT batch buffers.
- HW will trigger Render DOP CG on semaphore wait unsuccessful by default and can be disabled if not desired by programming Register Poll Mode Semaphore Wait Event IDLE message Disable bit in INSTPM register. Note that Render DOP CG will not be triggered on register semaphore wait un-successfull from INDIRECT_CTX pointer or BB_PER_CTX_PTR buffers.

Programming Notes

MI_SEMAPHORE_WAIT command must not be used in the middle of a tile pass on the posh pipe.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table>
<tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:23</td><td><b>MI Command Opcode</b>
<table>
<tr><td>Default Value:</td><td>1Ch MI_SEMAPHORE_WAIT</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>22</td><td><b>Memory Type</b>
This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit <i>must</i> be 1 if the <b>Per Process GTT Enable</b> bit is clear.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Per Process Graphics Address</td><td></td></tr>
<tr><td>1h</td><td>Global Graphics Address</td><td>This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.</td></tr>
</table></td></tr>
<tr><td></td><td>21:19</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>18</td><td><b>Workload Partition ID Offset Enable</b>
<table>
<tr><th>Description</th></tr>
<tr><td>This bit controls the memory read address computation for fetching the dat value from the memory for semaphore comparison. The final memory read address is computed by adding the Workload Partition ID" times the Address Offset to the memory address mentioned in the command. Workload Partition ID gets programmed through WPARID register and the Address Offset gets programmed through CS_MI_ADDRESS_OFFSET register.
Example: Final Memory Read Address[47:2] = (Workload Partition ID * Address Offset) + Memory Read Address [47:2]</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td></td><td>There is no offset added to the memory read address.</td></tr>
<tr><td>1</td><td></td><td>The final memory address is computed based on the Virtual Engine ID.</td></tr>
</table></td></tr>
<tr><td></td><td>17</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>16</td><td><b>Register Poll Mode</b>
This field control the semaphore wait behavior of polling from memory vs MMIO register.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1h</td><td>Register Poll <b>[Default]</b></td><td>In this mode HW periodically reads the semaphore data from MMIO register instead of memory for comparison until the condition is satisfied. Periodicity will be mentioned in a SEMA_WAIT_POLL register.
When operating in register poll mode, DW2 Semaphore Address (bits</td></tr>
</table></td></tr>
</table>

		22:2) carries the register MMIO offset to be polled. In register poll mode Memory Type field of this command are ignored by HW.
0h	Memory Poll	In this mode HW will functional as in regular mode and checks for semaphore data in memory.

Programming Notes
In register poll mode of operation of MI_SEMAPHORE_WAIT command, context switch is not supported on un-successful wait. Wait Mode must be always set to Polling Mode when Register Poll Mode is enabled. Preemption is supported on unsuccessful semaphore wait in Register Poll mode if operation.

15 **Wait Mode**

This bit specifies the WAIT behavior when the semaphore comparison fails and before the context is switched out.

Value	Name	Description
1h	Polling Mode	In this mode HW periodically reads the semaphore data from memory for comparison until it is context switched out. Periodicity will be mentioned in a SEMA_WAIT_POLL register.
0h	Signal Mode	[] In this mode HW will reacquire the semaphore data from memory for evaluating semaphore wait condition on receiving SIGNAL. Scheduler or SW can generate a SIGNAL to an engine by writing a value 0xFFFF_FFFF to the engines corresponding SEMAPHORE_SIGNAL_PORT register.

Programming Notes
Wait Mode must be always set to Polling Mode when Register Poll Mode is enabled.

14:12 **Compare Operation**

This field specifies the operation that will be executed to create the result that will either allow the context to continue or wait.

SAD = Semaphore Address Data
SDD = Semaphore Data Dword

Value	Name	Description
0h	SAD_GREATER_THAN_SDD	If Indirect fetched data is greater than inline data then continue.
1h	SAD_GREATER_THAN_OR_EQUAL_SDD	If Indirect fetched data is greater than or equal to inline data then continue.
2h	SAD_LESS_THAN_SDD	If Indirect fetched data is less than inline data then continue.
3h	SAD_LESS_THAN_OR_EQUAL_SDD	If Indirect fetched data is less than or equal to inline data then continue.

MI_SEMAPHORE_WAIT

		4h	SAD_EQUAL_SDD	If Indirect fetched data is equal to inline data then continue.
		5h	SAD_NOT_EQUAL_SDD	If Indirect fetched data is not equal to inline data then continue.
		6h	Reserved	
		7h	Reserved	
	11:10	**Reserved**		
		Access:	RO	
		Format:	MBZ	
	9:8	**Reserved**		
		Access:	RO	
		Format:	MBZ	
	7:0	**DWord Length**		
		Format:	=n	
		Value	**Name**	
		3h	Excludes DWord (0,1) **[Default]**	
1	31:0	**Semaphore Data Dword**		
		Format:	U32	
		This Data dword is supplied by software to control execution of the command buffer. This value is used as part of the comparison to result in waiting or continuing in the command parser if enabled.		
2..3	63:2	**Semaphore Address**		
		Format:	VIRTUAL_ADDR[63:2]	
		Register Poll Mode: In Register Poll mode of operation, Bits 22:2 (Bits 63:23 are reserved MBZ, HW enforced) specify the MMIO offset of the register for the semaphore. **Non Register Poll Mode:** This field is the Graphics Memory Address of the 32-bit value for the semaphore. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form.		
	1:0	**Reserved**		
		Access:	RO	
		Format:	MBZ	
4	31:22	**Reserved**		
		Access:	RO	
		Format:	MBZ	
	21:10	**Reserved**		
		Access:	RO	
		Format:	MBZ	

<table>
<tr><th colspan="3">MI_SEMAPHORE_WAIT</th></tr>
<tr><td rowspan="2"></td><td>9:2</td><td><b>Wait Token Number</b>
When context is switched out due to Semaphore wait, WaitTokenNumber is reported as Wait Detail in the CSB structure.</td></tr>
<tr><td>1:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

MI_SET_CONTEXT

MI_SET_CONTEXT

Source:	RenderCS
Length Bias:	2

The MI_SET_CONTEXT command is used to specify the logical context associated with the hardware context. A logical context is an area in memory used to store hardware context information, and the context is referenced via a 2KB-aligned pointer. If the (new) logical context is different (i.e., at a different memory address), the device saves the current HW context values to the current logical context address, and then restores (loads) the new logical context by reading the context from the new address and loading it into the hardware context state. If the logical context address specified in this command matches the current logical context address, this command is effectively treated as a NOOP. Specific to the Render command stream only. This command also includes some controls over the context save/restore process. The Force Restore bit can be used to refresh the on-chip device state from the same memory address if the indirect state buffers have been modified. The Restore Inhibit bit can be used to prevent the new context from being loaded at all. This must be used to prevent an uninitialized context from being loaded. Once software has initialized a context (by setting all state variables to initial values via commands), the context can then be stored and restored normally. When switching from a generic media context to a 3D context, the generic media state must be cleared via the Generic Media State Clear bit 16 in PIPE_CONTROL (or bit 4 in MI_FLUSH) before saving 3D context. MI_SET_CONTEXT commands are permitted only within a ring buffer (not within a batch buffer).
All context is saved and restored from a GGTT space.
This command does not initiate any interrupt due to context switch of any kind and does not support any workaround batch buffer or indirect context offset feature.

Programming Notes
For ring buffer mode, the first 128B(2 cache lines) of the context image are saved as zeros.
In execution list mode, this command must be preceded with a MI_ARB_ON_OFF command to disable arbitration and followed by a MI_ARB_ON_OFF command to enable arbitration. The first 320 bytes(5 cache lines) of the context image will be saved as zeros.
Arbitration Mode must be set to not allow lite restore prior to this command being executed. This bit is a field in the MI_ARB_ON_OFF command when in Execution list Mode.
This command needs to be always followed by a single MI_NOOP instruction to workaround a silicon issue.
MI_ARB_ON_OFF with 'Arbitration Enable Reset' set should be programmed before an MI_SET_CONTEXT command. MI_ARB_ON_OFF with 'Arbitration Enable' set should be programmed after an MI_SET_CONTEXT command.
MI_SET_CONTEXT command must not be programmed for a POSH enabled context.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 18h MI_SET_CONTEXT Format: OpCode

MI_SET_CONTEXT

	22:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 0h Format: =n
1	31:12	**Logical Context Address** Format: GraphicsAddress[31:12] This field contains the 4KB-aligned graphics memory address of the Logical Context that is to be loaded into the hardware context. If this address is equal to the CCID register associated with the current ring, no load will occur. Prior to loading this new context, the device will save the existing context as required. After the context switch operation completes, this address will be loaded into the associated CCID register. This field needs to be 4KB aligned virtual address.
	11:9	**Reserved** Access: RO Format: MBZ
	8	**Reserved, Must be 1** Format: MBO
	7:5	**Reserved** Access: RO Format: MBZ
	4	**Core Mode Enable** Format: Enable If set the Context Image will be offset based off the Core ID: If Core ID 0, no offset If Core ID 1, 36KB Offset
	3	**Resource Streamer State Save Enable** Format: Enable If set, the resource streamer state identified in the Logical Context Data section of the Memory Data Formats chapter is saved as part of switching away from this logical context. This bit will be stored in the associated CCID register to control the context save operation when switching away from this context (as part of a subsequent MI_SET_CONTEXT command).
	2	**Resource Streamer State Restore Enable** Format: Enable If set, the resource streamer state identified in the Logical Context Data section of the Memory Data Formats chapter is loaded (or restored) as part of switching to this logical context. This bit affects the switch (if required) to the context specified in Logical Context Address. This bit will also be stored in the associated CCID register to control a subsequent context save operation

MI_SET_CONTEXT

		when switching to this context (as part of a subsequent ring buffer switch).
	1	**Force Restore** When switching to this logical context a comparison between Logical Context Address and the contests of the CCID register is performed. Normally, matching addresses prevent a context restore from occurring; however, when this bit is set a context restore is forced to occur. This bit cannot be set with Restore Inhibit. Note: This bit is not saved in the associated CCID register. It only affects the processing of this command.
	0	**Restore Inhibit** If set, the restore of the HW context from the logical context specified by Logical Context Address is inhibited (i.e., the existing HW context values are maintained). This bit must be used to prevent the loading of an uninitialized logical context. If clear, the context switch proceeds normally. This bit cannot be set with Force Restore. Note: This bit is not saved in the associated CCID register. It only affects the processing of this command.

MI_SET_PREDICATE

MI_SET_PREDICATE

Source:	CommandStreamer
Length Bias:	1

Description

This command provides a mechanism to NOOP a section of commands programmed in the command buffer. This command on execution evaluates the predication status based on the following predication conditions enabled.

- Predicate Enable
- "Predicate Enable WPARID"

Predication status gets set if any of the above fields satisfy the predicate condition. On predicate status set, HW NOOPS the subsequent commands parsed until the predicate status is re-evaluated and reset on executing next MI_SET_PREDICATE command. The following commands can be programmed with "Predication Enable" bit field "to-be" or "not-to-be" predicated as part of the predication flow enforced by MI_SET_PREDICATE command.

MI_BATCH_BUFFER_START
MI_BATCH_BUFFER_END
MI_CONDITIONAL_BATCH_BUFFER_END
MI_SET_PREDICATE command will always get executed by HW irrespective of the predication status.
MI_SET_PREDCIATE commands predication status is context save/restored through MMIO register MI_SET_PREDICATE_RESULT to retain its functionality across the context switches. Predication based of MI_SET_PREDICATE_RESULT is only applied to the commands that are executed from Ring Buffer and Batch Buffer and doesn't apply to any other sources (context restore, Work Around Batch Buffers) of commands.

Programming Notes

- MI_SET_PREDICATE predication scope must be confined to commands programmed within a Batch Buffer (May include Nested and Chained Batch Buffers)..
- MI_SET_PREDICATE with Predicate Enable Must always have a corresponding MI_SET_PREDICATE with Predicate Disable within the same Batch Buffer.

DWord	Bit	Description	
0	31:29	**Command Type**	
		Default Value:	0h MI_COMMAND
		Format:	OpCode
	28:23	**MI Command Opcode**	
		Default Value:	01h MI_SET_PREDICATE
		Format:	OpCode
	22:6	**Reserved**	
		Access:	RO
		Format:	MBZ

MI_SET_PREDICATE

5:4

Predicate Enable WPARID

This field enables the predication based on the outcome of value resulting in bitwise AND of the bits in the WPARID and the PREDICATION_MASK Mask Register. WPARID and PREDICATION_MASK are non-privileged registers and context save/restored on a context switch.

Value	Name	Description
00h	NOOP Never	The predication status due to this field doesn't contribute to the overall predication status of MI_SET_PREDICATE command.
1h	NOOP on Zero Value	Predicate if (WPARID AND PREDICATE_MASK) == 0
2h	NOOP on Non-Zero Value	Predicate if (WPARID AND PREDICATE_MASK) != 0
3h	Reserved	

3:0

Predicate Enable

Source:	RenderCS, PositionCS, ComputeCS

Value	Name	Description
0h	NOOP Never	Predication is Disabled and CS will process commands as usual.
1h	NOOP on Result2 clear	Following Commands will be NOOPED by CS only if the MI_PREDICATE_RESULT_2 is clear.
2h	NOOP on Result2 set	Following Commands will be NOOPED by CS only if the MI_PREDICATE_RESULT_2 is set.
3h	NOOP on Result clear	Following Commands will be NOOPED by CS only if the MI_PREDICATE_RESULT is clear.
4h	NOOP on Result set	Following Commands will be NOOPED by CS only if the MI_PREDICATE_RESULT is set.
5h, 6h, 7h, 8h, 9h, Ah	Reserved	
Bh, Ch, Dh, Eh	Reserved	
Fh	NOOP Always	Following Commands will be NOOPED by RCS unconditionally.

3:0

Predicate Enable

Source:	BlitterCS, VideoCS, VideoEnhancementCS

Value	Name	Description
0h	Predicate Disable	Predication is Disabled and CS will process commands as usual.
1h	NOOP on Result2 Clear	Following Commands will be NOOPED by CS only if the MI_PREDICATE_RESULT_2 is clear.

MI_SET_PREDICATE

		2h	NOOP on Result2 Set	Following Commands will be NOOPED by CS only if the MI_PREDICATE_RESULT_2 is set.
		3h,4h,5h, 6h, 7h, 8h, 9h, Ah, Bh, Ch, Dh, Eh	Reserved	
		Fh	NOOP Always	Following Commands will be NOOPED by CS unconditionally.

MI_STORE_DATA_IMM

MI_STORE_DATA_IMM

Source:	CommandStreamer
Length Bias:	2

The MI_STORE_DATA_IMM command requests a write of the QWord constant supplied in the packet to the specified Memory Address. As the write targets a System Memory Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

This command supports writing to multiple consecutive dwords or qwords memory locations from the starting address.

Programming Notes

- This command should not be used within a "non-privilege" batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.
- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll un-cached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete eventually, there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 20h MI_STORE_DATA_IMM Format: OpCode
	22	**Use Global GTT** Format: Boolean If set, this command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit must be '1' if the Per Process GTT Enable bit is clear.
	21	**Store Qword** Format: Boolean If set, this command generates Qword writes to memory, two "Data Dword" are paired to form a Qword. Number of qwords generated depends upon the number of "Data Dword" programmed in the command. If 'x' number of "Data Dwords" are programmed in this command it results in "x/2" qword writes to memory. If reset this command generates Dwords writes to memory.

<table>
<tr><th colspan="3">MI_STORE_DATA_IMM</th></tr>
<tr><td></td><td></td><td>Number of dwords generated depends upon the number of "Data Dword" programmed in the command. If 'x' number of "Data Dwords" are programmed in this command it results in "x" dword writes to memory.</td></tr>
<tr><td></td><td>20:13</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>12</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>11</td><td><b>Workload Partition ID Offset Enable</b>
<table>
<tr><th>Description</th></tr>
<tr><td>This bit controls the memory write address computation for the store data update. The final memory write address is computed by adding the Workload Partition ID times the Address Offset to the memory address mentioned in the command. Workload Partition ID gets programmed in the WPARID register and the Address Offset gets programmed through CS_MI_ADDRESS_OFFSET register.
Example for store dword/qword: {Final Memory Write Address[47:2], ‘b00} = (Workload Partition ID* "Address Offset" + {Memory Write Address [47:2],‘b00}</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td></td><td>The final memory address is computed based on the Virtual Engine ID.</td></tr>
<tr><td>0</td><td></td><td>There is no offset added to the memory write address.</td></tr>
</table></td></tr>
<tr><td></td><td>10</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>9:0</td><td><b>DWord Length</b>
<table>
<tr><td>Format:</td><td>=n</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>2h</td><td>Store Dword <b>[Default]</b></td></tr>
<tr><td>3h</td><td>Store Qword</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>DWord Length programmed must not exceed 0x3FE.</td></tr>
<tr><td>If RS is enabled in the batch buffer, then the value of this field must not exceed 0x3F.</td></tr>
</table></td></tr>
<tr><td>1..2</td><td>63:2</td><td><b>Address</b>
<table>
<tr><td>Format:</td><td>VIRTUAL_ADDR[63:2]</td></tr>
</table>
GraphicsAddress is 64-bit value [63:0], but only a portion of it is used by hardware. The uppermost bits are ignored and MBZ. This field specifies Bits 47:2 of the Address where the DWord will be stored. As the store address must be DWord-aligned, Bits 1:0 of that address MBZ. This address must be 8B aligned for a store "QW" command.</td></tr>
</table>

MI_STORE_DATA_IMM		
	1	**Reserved** Access: RO Format: MBZ
	0	**Core Mode Enable** Format: U1 This bit is set then the address will be offset by the Core ID: If Core ID 0, then there is no offset If Core ID 1, then the Memory is offset by the size of the data(32b or 64b based off number of DW length).
3	31:0	**Data DWord 0** Format: U32 This field specifies the DWord value to be written to the targeted location. For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
4	31:0	**Data DWord 1** Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX

Source:	CommandStreamer
Length Bias:	2

The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes

- Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED.
- This command can be used for general software synchronization through variables in cacheable memory(i.e., where software does not need to poll uncached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete eventually, there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 21h MI_STORE_DATA_INDEX Format: OpCode
	22	**Reserved** Access: RO Format: MBZ
	21	**Use Per-Process Hardware Status Page** If this bit is set, this command will index into the per-process hardware status page at offset 0K from the LRCA. If clear, the Global Hardware Status Page will be indexed.
	20:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 1h Format: =n

<table>
<tr><th colspan="3">MI_STORE_DATA_INDEX</th></tr>
<tr><td>1</td><td>31:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:2</td><td><b>Offset</b><table><tr><td>Format:</td><td>U10 zero-based DWord offset into the HW status page.</td></tr></table>This field specifies the offset (into the hardware status page) to which the data will be written. Note that the first few DWords of this status page are reserved for special-purpose data storage - targeting these reserved locations via this command is UNDEFINED. This address must be 8B aligned for a store QW command.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[16, 1023]</td><td></td></tr></table></td></tr>
<tr><td></td><td>1:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>2</td><td>31:0</td><td><b>Data DWord 0</b><table><tr><td>Format:</td><td>U32</td></tr></table>This field specifies the DWord value to be written to the targeted location. For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).</td></tr>
<tr><td>3</td><td>31:0</td><td><b>Data DWord 1</b><table><tr><td>Format:</td><td>U32</td></tr></table>This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).</td></tr>
</table>

MI_STORE_REGISTER_MEM

MI_STORE_REGISTER_MEM

Source:	CommandStreamer
Length Bias:	2

The MI_STORE_REGISTER_MEM command requests a register read from a specified memory mapped register location in the device and store of that DWord to memory. The register address is specified along with the command to perform the read.

Programming Notes

- The command temporarily halts command execution.
- The memory address for the write is snooped on the host bus.
- This command should not be used from within a "non-privilege" batch buffer to access global virtual space. doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or "privilege" batch buffers to access global virtual space.
- This command will cause undefined data to be written to memory if given register addresses for the PGTBL_CTL_0 or FENCE registers.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 24h MI_STORE_REGISTER_MEM Format: OpCode
	22	**Use Global GTT** Format: Boolean It is allowed for this bit to be set when executing this command from a privileged (secure) batch or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	21	**Predicate Enable** Source: RenderCS, PositionCS, ComputeCS If set, this command is executed (or not) depending on the current value of the MI_PREDICATE internal state bit in MMIO register MI_PREDICATE_RESULT[0]. This command is ignored if PredicateEnable is set and value in the MMIO register MI_PREDICATE_RESULT[0] is 0. This command may also be dropped when MI_SET_PREDICATE condition to drop is true.

<table>
<tr><th colspan="3">MI_STORE_REGISTER_MEM</th></tr>
<tr><td></td><td>20</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>19</td><td><b>Add CS MMIO Start Offset</b>
This bit controls the functionality of the Register Address field in the command.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td></td><td>Register Address field in the command is treated as an offset from the executing Command Streamers MMIO start offset. Bits [22:2] of the Register Address are considered as dword offset to be added to the MMIO start offset of the corresponding command streamer.
Example: MI_STORE_REGISTER_MEM, ADD_CS_MMIO_START_OFFSET: true, Memory Address:0xABCD, Register Address: 0x1C_0030
The above command when executed on RenderCS will result in updating the memory address with the content of the MMIO offset 0x1C_2030 (0x00_2000 + 0x1C_0030) instead to 0x1C_0030. Note that RenderCS MMIO start offset is 0x2000.</td></tr>
<tr><td>0</td><td><b>[Default]</b></td><td>Register Address field in the command is absolute and not an offset from the executing command streamer MMIO start offset.</td></tr>
</table></td></tr>
<tr><td></td><td>18</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>17</td><td><b>MMIO Remap Enable</b>
This bit provides a mechanism in HW to remap the MMIO address in the MI command to the engine instance on which the command is getting executed, remapping in HW is done using engine specific remap table. Render and Compute engine share a common remapping table to facilitate remapping across engines, whereas a dedicated remap table for each of Video Decode and Video Enhancement engine class.
A MMIO remapping table per engine class is created with MMIO address belonging to multiple instances of an engine within an engine class. However Render and Compute engine share a common remapping table to facilitate remapping across engines, where as a dedicated remap table for each of Video Decode and Video Enhancement engine class.
This mode provides mechanism for SW to always use MMIO address belonging to fixed instance (instance zero) with in an engine class during command buffer creation agnostic to the instance on which it will get scheduled. This will also allow context interoperability across instances with in an engine class and extends to across engines in case of Render and Compute.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td></td><td>MMIO remapping will be applied to the MMIO address prior to using for any other functionality of the command.</td></tr>
<tr><td>0</td><td></td><td>MMIO remapping will not be applied to the MMIO address.</td></tr>
</table></td></tr>
</table>

MI_STORE_REGISTER_MEM

Programming Notes
• SW must always use MMIO address belonging to Instance-0 of an engine while enabling "MMIO Remap" in MI commands. • MMIO Remapping will be done by HW prior to doing any other functionality associated with the MI command or the privilege checks. • "Add CS MMIO Start Offset" must not be enabled when "MMIO Remap" is Enabled and Vice-versa. • When remapping is not found in the remap table, HW will use the MMIO address directly without any modification.

16 **Workload Partition ID Offset Enable**

Description
This bit controls the memory write address computation for the store data update. The final memory write address is computed by adding the Workload Partition ID times the "Address Offset" to the memory address mentioned in the command. Workload Partition ID gets programmed in the WPARID register and the Address Offset gets programmed through CS_MI_ADDRESS_OFFSET register. Example: {Final Memory Write Address[47:2], 2'b00} = (Workload Partition ID* Address Offset) + {Memory Write Address [47:2], 2'b00}

Value	Name	Description
1		The final memory address is computed based on the Virtual Engine ID.
0		There is no offset added to the memory write address.

15:8 **Reserved**

Access:	RO
Format:	MBZ

7:0 **DWord Length**

Default Value:	2h Excludes DWord (0,1)
Format:	=n

1 31:23 **Reserved**

Access:	RO
Format:	MBZ

22:2 **Register Address**

Format:	MMIOAddress[22:2]

This field specifies Bits 22:2 of the Register offset the DWord will be read from. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ.

MI_STORE_REGISTER_MEM

		Programming Notes • Storing a VGA register is not permitted and will store an UNDEFINED value. • The values of PGTBL_CTL0 or any of the FENCE registers cannot be stored to memory; UNDEFINED values will be written to memory if the addresses of these registers are specified.
	1:0	**Reserved** Access: RO Format: MBZ
2..3	63:2	**Memory Address** Format: GraphicsAddress[63:2] This field specifies the address of the memory location where the register value specified in the DWord above will be written. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[63:2] for a DWord register GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	1:0	**Reserved** Access: RO Format: MBZ

MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH

Source:	BlitterCS
Length Bias:	1

Blocks PM Flush Requests.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH Format: OpCode
	22:1	**Reserved** Access: RO Format: MBZ
	0	**Suspend Flush** Format: Enable This field suspends flush due to a PM flush request.

MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH		
Source:	RenderCS	
Length Bias:	1	
Blocks PM Flush Requests.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH Format: OpCode
	22:1	**Reserved** Access: RO Format: MBZ
	0	**Suspend Flush** Format: Enable This field suspends flush due to a PM flush request.

MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH		
Source:	VideoCS	
Length Bias:	1	
Blocks PM Flush Requests.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH Format: OpCode
	22:1	**Reserved** Access: RO Format: MBZ
	0	**Suspend Flush** Format: Enable This field suspends flush due to a PM flush request.

MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH		
Source:	VideoEnhancementCS	
Length Bias:	1	
Blocks PM Flush Requests.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH Format: OpCode
	22:1	**Reserved** Access: RO Format: MBZ
	0	**Suspend Flush** Format: Enable This field suspends flush due to a PM flush request.

MI_TOPOLOGY_FILTER

MI_TOPOLOGY_FILTER

Source:	RenderCS
Length Bias:	1

This command is used to specify a specific 3DPrimType value, where the CS will ignore all 3DPRIMITIVE commands that do no have a matching 3DPrimType. This primitive culling is optional (turned off by using this command with a Topology Filter Value of 0). **This command is specific to the Render command stream only.**

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Dh MI_TOPOLOGY_FILTER Format: OpCode
	22:6	**Reserved** Access: RO Format: MBZ
	5:0	**Topology Filter Value** Format: 3D_Prim_Topo_Type When non-zero, the CS will discard all 3DPRIMITIVE commands which do not match the specified 3DPrimTopologyType. When zero, no filtering is performed (normal operation).

MI_UPDATE_GTT

MI_UPDATE_GTT

Source:	BSpec
Length Bias:	2

The MI_UPDATE_GTT command is used to update GTT page table entries in a coherent manner and at a predictable place in the command flow.
A PIPE_CONTROL flush command with "CS Stall" bit set must be programmed prior to MI_UPDATE_GTT command, since work associated with preceding commands that are still in the pipeline may be referencing GTT entries that will be changed by its execution. The flush must also invalidate TLBs and read caches that may become invalid as a result of the changed GTT entries. A PIPE_CONTROL flush command with "CS Stall" bit set must be programmed post MI_UPDATE_GTT command to ensure the GGTT is updated with modified page table entries before the following workload references the modified entries.
PIPE_CONTROL flush is not required if it can be guaranteed that the pipeline is free of any work that relies on changing GTT entries (such as MI_UPDATE_GTT contained in a paging DMA buffer that is doing only update/mapping activities and no rendering).
MI_UPDTE_GTT command is privilege operation and will be converted to a no-op and an error flagged if it is executed from within a non-secure batch buffer.
PPGTT updates cannot be done via **MI_UPDATE_GTT**, gfx driver will have to use MI_STORE_DATA_IMM for PPGTT inline updates.

The MI_UPDATE_GTT command is used to update GGTT page table entries in a coherent manner and at a predictable place in the command flow. A MI_FLUSH_DWORD flush command with "CS Stall" bit set must be programmed prior to MI_UPDATE_GTT command, since work associated with preceding commands that are still in the pipeline may be referencing GTT entries that will be changed by its execution. The flush must also invalidate TLBs and read caches that may become invalid as a result of the changed GTT entries. A MI_FLUSH_DWORD flush command with "CS Stall" bit set must be programmed post MI_UPDATE_GTT command to ensure the GGTT is updated with modified page table entries before the following workload references the modified entries. MI_FLUSH_DWORD flush is not required if it can be guaranteed that the pipeline is free of any work that relies on changing GTT entries (such as MI_UPDATE_GTT contained in a paging DMA buffer that is doing only update/mapping activities and no rendering). MI_UPDATE_GTT command is privilege operation and will be converted to a no-op and an error flagged if it is executed from within a non-secure batch buffer. PPGTT updates cannot be done via **MI_UPDATE_GTT** , gfx driver will have to use MI_STORE_DATA_IMM for PPGTT inline updates.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 23h MI_UPDATE_GTT Format: OpCode
	22:10	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">MI_UPDATE_GTT</th></tr>
<tr><td></td><td>9:0</td><td><b>DWord Length</b>
<table><tr><td>Format:</td><td>=n</td></tr></table>
n = 2b (where b = # of Entry Data included)
<table><tr><th>Value</th><th>Name</th></tr><tr><td>[2,1022]</td><td></td></tr><tr><td>2</td><td><b>[Default]</b></td></tr></table></td></tr>
<tr><td>1</td><td>31:12</td><td><b>Entry Address</b>
<table><tr><td>Format:</td><td>GraphicsAddress[31:12]</td></tr></table>
This field holds the QW offset of the first table entry to be modified in GGTT.</td></tr>
<tr><td></td><td>11:0</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>2..n</td><td>63:0</td><td><b>Entry Data</b>
<table><tr><td>Format:</td><td>U64</td></tr></table>
This Dword becomes the new page table entry. See PPGTT/Global GTT Table Entries (PTEs) in Memory Interface Registers.</td></tr>
</table>

MI_USER_INTERRUPT

MI_USER_INTERRUPT

Source:	BlitterCS
Length Bias:	1

The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	**Reserved** Access: RO Format: MBZ

MI_USER_INTERRUPT

MI_USER_INTERRUPT

Source:	RenderCS
Length Bias:	1

The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	**Reserved** Access: RO Format: MBZ

MI_USER_INTERRUPT

MI_USER_INTERRUPT

Source:	VideoCS
Length Bias:	1

The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	**Reserved** Access: RO Format: MBZ

MI_USER_INTERRUPT

MI_USER_INTERRUPT

Source:	VideoEnhancementCS
Length Bias:	1

The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	**Reserved** Access: RO Format: MBZ

MI_WAIT_FOR_EVENT_2

MI_WAIT_FOR_EVENT_2

Source:	RenderCS, BlitterCS
Length Bias:	1

The MI_WAIT_FOR_EVENT_2 command is used to pause command stream processing of an engine (render or blitter) until a specific display event occurs (V-Blank) or a specific condition (Flip Pending, Scanline Pending) exists.

- Display engine can be configured to generate periodic V-Blank event to an engine.
- An engine on executing MI_LOAD_SCANLINE_INCL/EXCL command for a display pipe, expects a corresponding scanline event response from display engine. Engine tracks the pending scanline response for each of the display pipe separately.
- An engine on executing MI_DISPLAY_FLIP command for a display plane, expects a corresponding flip done event response from display engine. Engine tracks the pending flip done response (flip pending) for each of the display plane separately. Display flip could be of type Sync Flip or Async Flip, hence engine also tracks the type of flip (Sync or Async) along with the pending flip done response for a given display plane.

Only one event or condition can be specified in the command -- specifying multiple events or conditions is UNDEFINED. The command parser will halt until the event occurs or condition exists on parsing this command. Note that if a specified condition (Pending Flip Done response or Pending Scanline response) does not exist at the time the parser executes this command, the parser proceeds, treating this command as a no-operation (Ex: Command is no-operation when parsed with Display Plane Flip Pending Wait Enable set to Display Plane-1 when there is no outstanding flip done response for Display Plane-1 in the engine).

Execution List Mode of Scheduling:

An engine on evaluating unsuccessful MI_WAIT_FOR_EVENT_2 (results in pausing command stream) triggers synchronous context switch stating the switch reason in Context Status Buffer. With exception of not triggering synchronous context switch on unsuccessful MI_WAIT_FOR_EVENT_2 due to Flip Pending on an Async flip. Note that synchronous context switch can be inhibited through programming Inhibit Synchronous Context Switch bit in CTXT_SR_CTL register or by disabling arbitration through MI_ARB_ON_OFF command around MI_WAIT_FOR_EVENT_2. When synchronous context switch is inhibited and the engine is waiting on an unsuccessful MI_WAIT_FOR_EVENT_2, a submission of new execlist will trigger preemption process switching out the context. With exception of not triggering preemption on an unsuccessful MI_WAIT_FOR_EVENT_2 due to Flip Pending on an Async flip.

Engine will always re-evaluate the wait condition for context switched out due to unsuccessful MI_WAIT_FOR_EVENT2 on resubmission of the context.

DWord	Bit	Description		
0	31:29	**Command Type**		
			Default Value:	0h MI_COMMAND
			Format:	OpCode
	28:23	**MI Command Opcode**		
			Default Value:	04h MI_WAIT_FOR_EVENT_2
			Format:	OpCode

MI_WAIT_FOR_EVENT_2

22:15 **Reserved**

Access:	RO
Format:	MBZ

14:12 **Display Pipe Scan Line Wait Enable**

Format:	Enable

This field enables a wait while a Display Pipe "Scan Line" condition exists. This condition is defined as the the start of the scan line specified in the Pipe B Display Scan Line Count Range Compare Register.

Value	Name
0h	No Wait
1h	Display Pipe A
2h	Display Pipe B
3h	Display Pipe C
4h	Display Pipe D
[5h,7h]	Reserved

11 **Reserved**

Access:	RO
Format:	MBZ

10:8 **Display Pipe Vertical Blank Wait Enable**

Format:	Enable

This field enables a wait until the next Display Pipe "Vertical Blank" event occurs. This event is described as the start of the next Display A vertical blank period. Note that this can cause a wait for up to an entire refresh period.

Value	Name
0h	No Wait
1h	Display Pipe A
2h	Display Pipe B
3h	Display Pipe C
4h	Display Pipe D
[5h,7h]	Reserved

7:6 **Reserved**

Access:	RO
Format:	MBZ

5:0 **Display Plane Flip Pending Wait Enable**

Format:	Enable

This field enables a wait for the duration of a Display Plane Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

MI_WAIT_FOR_EVENT_2

Value	Name
0h	No Wait
1h	Display Plane-1
2h	Display Plane-2
3h	Display Plane-3
4h	Display Plane-4
5h	Display Plane-5
6h	Display Plane-6
7h	Display Plane-7
8h	Display Plane-8
9h	Display Plane-9
Ah	Display Plane-10
Bh	Display Plane-11
Ch	Display Plane-12
Dh	Display Plane-13
Eh	Display Plane-14
Fh	Display Plane-15
10h	Display Plane-16
11h	Display Plane-17
12h	Display Plane-18
13h	Display Plane-19
14h	Display Plane-20
15h	Display Plane-21
16h	Display Plane-22
17h	Display Plane-23
18h	Display Plane-24
19h	Display Plane-25
1Ah	Display Plane-26
1Bh	Display Plane-27
1Ch	Display Plane-28
1Dh	Display Plane-29
1Eh	Display Plane-30
1Fh	Display Plane-31
20h	Display Plane-32
[21h, 3Fh]	Reserved

MI_WAIT_FOR_EVENT

MI_WAIT_FOR_EVENT

Source:	RenderCS, BlitterCS
Length Bias:	1

The MI_WAIT_FOR_EVENT command is used to pause command stream processing of an engine (render or blitter) until a specific display event occurs (V-Blank) or a specific condition (Flip Pending, Scanline Pending) exists.

- Display engine can be configured to generate periodic V-Blank event to an engine.
- An engine on executing MI_LOAD_SCANLINE_INCL/EXCL command for a display pipe, expects a corresponding scanline event response from display engine. Engine tracks the pending scanline response for each of the display pipe separately.
- An engine on executing MI_DISPLAY_FLIP command for a display plane, expects a corresponding flip done event response from display engine. Engine tracks the pending flip done response (flip pending) for each of the display plane separately. Display flip could be of type Sync Flip or Async Flip, hence engine also tracks the type of flip (Sync or Async) along with the pending flip done response for a given display plane.

Only one event or condition can be specified in the command -- specifying multiple events or conditions is UNDEFINED. The command parser will halt until the event occurs or condition exists on parsing this command. Note that if a specified condition (Pending Flip Done response or Pending Scanline response) does not exist at the time the parser executes this command, the parser proceeds, treating this command as a no-operation (Ex: Command is no-operation when parsed with Display Plane Flip Pending Wait Enable set to Display Plane-1 when there is no outstanding flip done response for Display Plane-1 in the engine).

Execution List Mode of Scheduling:

An engine on evaluating unsuccessful MI_WAIT_FOR_EVENT(results in pausing command stream) triggers synchronous context switch stating the switch reason in Context Status Buffer. With exception of not triggering synchronous context switch on unsuccessful MI_WAIT_FOR_EVENT due to Flip Pending on an Async flip. Note that synchronous context switch can be inhibited through programming Inhibit Synchronous Context Switch bit in CTXT_SR_CTL register or by disabling arbitration through MI_ARB_ON_OFF command around MI_WAIT_FOR_EVENT. When synchronous context switch is inhibited and the engine is waiting on an unsuccessful MI_WAIT_FOR_EVENT, a submission of new execlist will trigger preemption process switching out the context. With exception of not triggering preemption on an unsuccessful MI_WAIT_FOR_EVENT due to Flip Pending on an Async flip.

Engine will always re-evaluate the wait condition for context switched out due to unsuccessful MI_WAIT_FOR_EVENT on resubmission of the context.

DWord	Bit	Description	
0	31:29	**Command Type**	
		Default Value:	0h MI_COMMAND
		Format:	OpCode
	28:23	**MI Command Opcode**	
		Default Value:	03h MI_WAIT_FOR_EVENT
		Format:	OpCode

MI_WAIT_FOR_EVENT		
	22	**Reserved** Access: RO Format: MBZ
	21	**Display Plane 1 C Vertical Blank Wait Enable** Format: Enable This field enables a wait until the next Display Plane 1 C "Vertical Blank" event occurs. This event is described as the start of the next Display C vertical blank period. Note that this can cause a wait for up to an entire refresh period. See Vertical Blank Event in the Device Programming Interface chapter of MI Functions.
	20	**Display Plane 6 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 2 C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	19	**Display Plane 12 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 4 C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	18	**Display Plane 11 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 4 B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	17	**Display Plane 10 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 4 A Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	16	**Display Plane 9 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 3 C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	15	**Display Plane 3 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 1 C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

MI_WAIT_FOR_EVENT

	14	**Display Plane 1 C Scan Line Wait Enable** Format: Enable This field enables a wait while a Display Plane 1 C "Scan Line" condition exists. This condition is defined as the the start of the scan line specified in the Pipe C Display Scan Line Count Range Compare Register.
	13:12	**Reserved** Access: RO Format: MBZ
	11	**Display Plane 1 B Vertical Blank Wait Enable** Format: Enable This field enables a wait until the next Display Plane 1 B "Vertical Blank" event occurs. This event is described as the start of the next Display B vertical blank period. Note that this can cause a wait for up to an entire refresh period.
	10	**Display Plane 5 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 2 B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	9	**Display Plane 2 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	8	**Display Plane 1 B Scan Line Wait Enable** Format: Enable This field enables a wait while a Display Plane 1 B "Scan Line" condition exists. This condition is defined as the the start of the scan line specified in the Pipe B Display Scan Line Count Range Compare Register.
	7	**Display Plane 8 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 3 B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	6	**Display Plane 7 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 3 A Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	5:4	**Reserved** Access: RO Format: MBZ

MI_WAIT_FOR_EVENT

	3	**Display Plane 1 A Vertical Blank Wait Enable** Format: Enable This field enables a wait until the next Display Plane 1 A "Vertical Blank" event occurs. This event is described as the start of the next Display A vertical blank period. Note that this can cause a wait for up to an entire refresh period.
	2	**Display Plane 4 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 2 A Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	1	**Display Plane 1 Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane 1 A Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	0	**Display Plnae 1 A Scan Line Wait Enable** Format: Enable This field enables a wait while a Display Plane 1 A "Scan Line" condition exists. This condition is defined as the the start of the scan line specified in the Pipe A Display Scan Line Count Range Compare Register.

Monitor Event

MSD_MONITOR_EVENT - Monitor Event

Source:	EuSubFunctionGateway
Length Bias:	1

Gateway will record for this thread if this Event ID is signaled.

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of GRF registers sent as the message payload. Value | Name | Description 1 | One **[Default]** | See MDP_EVENT Event Data Payload definition.
	24:20	**Response Length** Default Value: 0 None Format: U5 Specifies the number of GRF registers expected as the message response payload.
	19:3	**Reserved** Access: RO Format: MBZ
	2:0	**Monitor Event Subfunction** Default Value: 0x2 Format: OpCode

Monitor No Event

<table>
<tr><th colspan="3">MSD_MONITOR_NO_EVENT - Monitor No Event</th></tr>
<tr><td colspan="3">Source: EuSubFunctionGateway
Length Bias: 1</td></tr>
<tr><td colspan="3">Gateway will stop recording any Events for this thread.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>28:25</td><td><b>Message Length</b>
<table><tr><td>Format:</td><td>U4</td></tr></table>
Specifies the number of GRF registers sent as the message payload.
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>1</td><td>One <b>[Default]</b></td><td>Data payload is ignored.</td></tr></table></td></tr>
<tr><td></td><td>24:20</td><td><b>Response Length</b>
<table><tr><td>Default Value:</td><td>0 None</td></tr><tr><td>Format:</td><td>U5</td></tr></table>
Specifies the number of GRF registers expected as the message response payload.</td></tr>
<tr><td></td><td>19:3</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>2:0</td><td><b>Monitor No Event Subfunction</b>
<table><tr><td>Default Value:</td><td>0x3</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
</table>

Move

mov - Move

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

The mov instruction moves the components in src0 into the channels of dst. If src0 and dst are of different types, format conversion is performed. If src0 is a scalar immediate, the immediate value is loaded into enabled channels of dst.
A mov with the same source and destination type, no source modifier, and no saturation is a raw move. i.e. the destination is written with an unmodified copy of the source. A packed byte destination region (B or UB type with HorzStride == 1 and ExecSize > 1) can only be written using raw move.

When denorm mode is flush to zero, a raw mov instruction with saturation modifier will not flush the denorm input or output to zero (Denorm is preserved).

Format:

```
 [(pred)] mov[.cmod] (exec_size) dst src0
```

Programming Notes
A *mov* instruction with a source modifier always copies a denorm source value to a denorm destination value(in the manner of a raw move).
There is no direct conversion from B/UB to DF or DF to B/UB. Use two instructions and a word or DWord intermediate type.
There is no direct conversion from B/UB to Q/UQ or Q/UQ to B/UB. Use two instructions and a word or DWord intermediate integer type.
There is no direct conversion from HF to DF or DF to HF. Use two instructions and F (Float) as an intermediate type.
There is no direct conversion from HF to Q/UQ or Q/UQ to HF. Use two instructions and F (Float) or a word integer type or a DWord integer type as an intermediate type.

Restriction
ALT mode is not honored by raw move.
IP register must not be used as destination operand when EU Fusion is enabled.
Float to Bfloat16 conversion must not use Predication, Conditional Modifiers, Saturation and Source Modifiers. Denorms are always retained. Rounding Mode RTNE is used in respect of programmed rounding mode.

Syntax

```
[(pred)] mov[.cmod] (exec_size) reg reg
 [(pred)] mov[.cmod] (exec_size) reg imm32
```

mov - Move

```
[(pred)] mov[.cmod] (exec_size) reg imm64
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = src0.chan[n];
     }
 }
```

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
*B,*W,*D	F
F	*B,*W,*D
F	F
*B,*W,*D	HF
F	HF
HF	*B,*W,*D
HF	F
HF	HF
F	BF

DWord	Bit	Description
0..3	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true)
	95:92	**CondCtrl** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **FlagModifier**
	95:64	**Src0.ImmValue[63:32]** Exists If: ([Src0.IsImm]==true) AND (([Src0.DataType]==:q) OR ([Src0.DataType]==:uq) OR ([Src0.DataType]==:df))
	87:84	**Src0.VertStride** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **VertStride**
	83:81	**Src0.Width** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **Width**

mov - Move

<table>
<tr><td></td><td>80</td><td><b>Src0.AddrMode</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))</td></tr><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))</td></tr><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
</table>

mov - Move

<table>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td><b>Saturate</b></td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td><b>AccWrCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no</td></tr>
</table>

mov - Move

<table>
<tr><td></td><td></td><td>predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table><tr><td>Format:</td><td>PredCtrl</td></tr></table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table><tr><td>Format:</td><td>ChanOff</td></tr></table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table><tr><td>Format:</td><td>ExecSize</td></tr></table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b>
<table><tr><td>Format:</td><td>Header</td></tr></table></td></tr>
</table>

Move Indexed

movi - Move Indexed

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	true
Source Modifier:	true

The movi instruction performs a fast component-wise indexed move for subfields from src0 to dst. The source operand must be an indirectly-addressed register. All channels of the source operand share the same register number, which is provided by the register field of the first address subregister, with a possible immediate register offset. The register fields of the subsequent address subregisters are ignored by hardware. The subregister number of a source channel is provided by the subregister field of the corresponding address subregister, with a possible immediate subregister offset.

The destination register may be either a directly-addressed or an indirectly-addressed register.

This instruction effectively performs a subfield shuffling from one register to another.

Format:

```
[(pred)] movi (exec_size) dst src0 src1
```

Programming Notes
The source register is calculated by adding the register portion of the first index register with the register portion of the address immediate, a0.0[11:5] + addr_imm[9:5] For byte movi, byte0 of the destination is selected by (a0.0[4:0]), byte1 is selected by (a0.1[4:0]), ..., and byte7 is selected by (a0.7[4:0]). The rest of the bytes are undefined. For word movi, byte0 of the destination is selected by {a0.0[4:1], 0}, byte1 is selected by {a0.0[4:1],1b}, byte2 is selected by {a0.1[4:1],0b}, byte3 is selected by {a0.1[4:1], 1b}, ..., and byte15is selected by {a0.7[4:1], 1b}. The rest of the bytes are undefined. For DWord or float movi, byte0 of the destination is selected by {a0.0[4:2], 00b}, byte1 is selected by {a0.0[4:2],01b}, byte2 is selected by {a0.0[4:2],10b}, byte3 is selected by {a0.0[4:2], 11b}, byte4 is selected by {a0.1[4:2], 00b}, byte5 is selected by {a0.1[4:2], 01b}, ..., byte31is selected by {a0.7[4:2],11b}. For all 3 conditions above, a0.n[4:0] = a0.n[4:0] + addr_imm[4:0].

Restriction
Source operand cannot be accumulators. The source operand must be a general register.
The source and destination must have the same type.
The address register for the source must be a0.0 or a0.8.
The destination register (directly or indirectly addressed) must be 16-byte aligned.
The destination region (directly or indirectly addressed) must point to the same GRF register.
The destination stride in bytes must equal the source element size in bytes.
All the index registers (address subregisters) used must point to the same GRF register.
The instruction must use 1x1 indirect regioning.

movi - Move Indexed

The destination offset is only used to create channel enables. Each element of the destination is directly mapped to the index registers for the movi instruction. i.e. a0.0 -> dst.0, a0.1 -> dst.1, a0.2 -> dst.2, etc.

Only 8 address subregisters are used (a0.0-a0.7 or a0.8-a0.15). Destination element will be sourced from address register (a0.0 or a0.8), for example:
movi (8) r31.0:uw r[a0.0,0]<1;1,0>:uw // r31.0:uw<-a0.0:uw, r31.1:uw<-a0.1:uw, etc.
movi (8) r31.0:uw r[a0.8,0]<1;1,0>:uw // r31.0:uw<-a0.8:uw, r31.1:uw<-a0.9:uw, etc.
movi (8) r31.8:uw r[a0.0,0]<1;1,0>:uw // r31.8:uw<-a0.0:uw, r31.9:uw<-a0.1:uw, etc.
movi (8) r31.8:uw r[a0.8,0]<1;1,0>:uw // r31.8:uw<-a0.8:uw, r31.9:uw<-a0.9:uw, etc.
movi (8) r31.0:ud r[a0.0,0]<1;1,0>:ud // r31.0:ud<-a0.0:ud, r31.1:ud<-a0.1:ud, etc.
movi (8) r31.0:ud r[a0.8,0]<1;1,0>:ud // r31.0:ud<-a0.8:ud, r31.1:ud<-a0.9:ud, etc.

Conditional Modifier is not allowed for this instruction.

Syntax

```
[(pred)] movi (exec_size) reg reg null
 [(pred)] movi (exec_size) reg reg imm32
```

Pseudocode

```
            Evaluate(WrEn);
            srcregfile = regfile(src0);
            imm_offset = (src1 == NULL) ? addr_imm : src1;
            srcregbase = reg(address[0]) + reg(imm_offset);
            for ( n = 0; n < RegWidth; n++ ) {
                if ( WrEn.chan[n] ) {
                    srcsubreg = subreg(address[n] + imm_offset);
                    dst.chan[n] = srcregfile.srcreg.srcsubreg;
                }
            }
```

pre>

Src Types	Dst Types
B	B
UB	UB
W	W
UW	UW
D	D
UD	UD

DWord	Bit	Description
0..3	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true)
	95:92	**CondCtrl** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **FlagModifier**

movi - Move Indexed

	95:64	**Src0.ImmValue[63:32]**	Exists If: ([Src0.IsImm]==true) AND (([Src0.DataType]==:q) OR ([Src0.DataType]==:uq) OR ([Src0.DataType]==:df))	
	87:84	**Src0.VertStride**	Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	Format: VertStride
	83:81	**Src0.Width**	Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	Format: Width
	80	**Src0.AddrMode**	Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	Format: AddrMode
	79:66	**Src0.Operand**	Exists If: (([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Direct)	Format: DirectOperand
	79:66	**Src0.Operand**	Exists If: (([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Indirect)	Format: IndirectOperand
	65:64	**Src0.HorzStride**	Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	Format: HorzStride
	63:50	**Dst.Operand**	Exists If: ([Dst.AddrMode]==Indirect)	Format: IndirectOperand
	63:50	**Dst.Operand**	Exists If: ([Dst.AddrMode]==Direct)	Format: DirectOperand
	49:48	**Dst.HorzStride**		Format: HorzStride

movi - Move Indexed

47 **Reserved**

Access:	RO
Format:	MBZ

46 **Src0.IsImm**

This field indicate that Source 0 operand is carrying an immediate value.

Value	Name
0	false **[Default]**
1	true

45:44 **Src0.Mod**

Format:	**SrcMod**

43:40 **Src0.DataType**

Exists If:	([Src0.IsImm]==false)
Format:	**RegDataType**

43:40 **Src0.DataType**

Exists If:	([Src0.IsImm]==true)
Format:	**ImmDataType**

39:36 **Dst.DataType**

Format:	**RegDataType**

35 **Dst.AddrMode**

Format:	**AddrMode**

34 **Saturate**

Format:	**Saturate**

33 **AccWrCtrl**

Format:	**AccWrCtrl**

32 **AtomicCtrl**

Format:	**AtomicCtrl**

31 **MaskCtrl**

Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".

Value	Name	Description
0	Normal **[Default]**	Normal. Per channel write enable used for final write enable generation.
1	NoMask	NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.

30 **Reserved**

movi - Move Indexed

29 **CmptCtrl**

Format:	MBZ

Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.

Value	Name	Description
0	NoCompaction **[Default]**	No compaction. 128-bit native instruction supporting all instruction options.
1	Compacted	Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.

28 **PredInv**

This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields

Value	Name	Description
0	Positive **[Default]**	Positive polarity of predication. Use the predication mask produced by PredCtrl.
1	Negative	Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.

27:24 **PredCtrl**

Format:	**PredCtrl**

This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.

23 **FlagRegNum[0]**

This field specifies bit[0] of the register number for a flag register operand.

22 **FlagSubRegNum**

This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.

21:19 **ChanOff**

Format:	**ChanOff**

This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.

movi - Move Indexed

	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Multiply

mul - Multiply

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

The mul instruction performs component-wise multiplication of src0 and src1 and stores the results in dst. When multiplying integer datatypes, if src0 is DW and src1 is W, irrespective of the destination datatype, the accumulator maintains full 48-bit precision. This is required to handle the macro for 32x32 multiplication. The macro described in the mach instruction should be used to obtain the full precision 64-bit multiplication results.

Note: A 32x32 multiply operation is handled natively, without a macro. When operating in this mode, the resulting 64-bit data is packed, unlike the macro, where the lower and upper 32 bits of the result are written to different general registers by two separate instructions. Refer to the macro description for details.

When multiplying integer data types, if one of the sources is a DW, the resulting full precision data is stored in the accumulator. However, if the destination data type is either W or DW, the low bits of the result are written to the destination register and the remaining high bits are discarded. This results in undefined Overflow and Sign flags. Therefore, conditional modifiers and saturation (.sat) cannot be used in this case.

Format:

```
        [(pred)] mul[.cmod] (exec_size) dst src0 src1
```

Floating-Point Addition of A (Column) and B (Row) in IEEE Mode

	-inf	**-finite**	**-1.0**	**-denorm**	**-0**	**+0**	**+denorm**	**+1.0**	**+finite**	**+inf**	**NaN**
-inf	+inf	+inf	+inf	NaN	NaN	NaN	NaN	-inf	-inf	-inf	NaN
-finite	+inf	*	-A	+0	+0	-0	-0	A	**	-inf	NaN
-1.0	+inf	-B	+1.0	+0	+0	-0	-0	-1.0	-B	-inf	NaN
-denorm	NaN	+0	+0	+0	+0	-0	-0	-0	-0	NaN	NaN
-0	NaN	+0	+0	+0	+0	-0	-0	-0	-0	NaN	NaN
+0	NaN	-0	-0	-0	-0	+0	+0	+0	+0	NaN	NaN
+denorm	NaN	-0	-0	-0	-0	+0	+0	+0	+0	NaN	NaN
+1.0	-inf	B	-1.0	-0	-0	+0	+0	+1.0	B	+inf	NaN
+finite	-inf	**	-A	-0	-0	+0	+0	A	*	+inf	NaN
+inf	-inf	-inf	-inf	NaN	NaN	NaN	NaN	+inf	+inf	+inf	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Notes:											
*	Result may be {–finite. +inf (overflow)}.										
**	Result may be {-inf (overflow, –finite}.										

Floating-Point Addition of A (Column) and B (Row) in ALT Mode

mul - Multiply

	-fmax	-finite	-1.0	-denorm	-0	+0	+denorm	+1.0	+finite	+fmax	***
-fmax	+fmax	+fmax	+fmax	-0	-0	+0	+0	-fmax	-fmax	-fmax	
-finite	+fmax	*	-A	+0	+0	-0	-0	A	**	-fmax	
-1.0	+fmax	-B	+1.0	+0	+0	-0	-0	-1.0	-B	-fmax	
-denorm	+0	+0	+0	+0	+0	-0	-0	-0	-0	-0	
-0	+0	+0	+0	+0	+0	-0	-0	-0	-0	-0	
+0	-0	-0	-0	-0	-0	+0	+0	+0	+0	+0	
+denorm	-0	-0	-0	-0	-0	+0	+0	+0	+0	+0	
+1.0	-fmax	B	-1.0	-0	-0	+0	+0	+1.0	B	+fmax	
+finite	-fmax	**	-A	-0	-0	+0	+0	A	*	+fmax	
+fmax	-fmax	-fmax	-fmax	-0	-0	+0	+0	+fmax	+fmax	+fmax	
*******											
Notes:											
*	Result may be {+finite, +fmax (overflow)}.										
**	Result may be {–fmax (overflow), –finite}.										
***	Result is undefined if A or B is {–inf, +inf, NaN}.										

Restriction
Integer source operands cannot be accumulators.
When multiplying a DW and any lower precision integer, the DW operand must on src0.
When multiplying a DW and any lower precision integer, source modifier is not supported.
When multiplying DW X DW, resulting dst can only be QW precision. If DW precision is required at output than MUL/MACH macro must be used.

Syntax

```
[(pred)] mul[.cmod] (exec_size) reg reg reg
 [(pred)] mul[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = src0.chan[n] * src1.chan[n];
     }
 }
```

Src Types	Dst Types
*B	*B
*B	*W
*B	*D

mul - Multiply

*W	*W
*W	*D
*W, *D	*D
F	F
HF	HF
BF, F	BF, F

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)
	125:122	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	121:120	**Src1.Mod** Exists If: ([Src1.IsImm]==false) Format: **SrcMod**
	119:116	**Src1.VertStride** Exists If: ([Src1.IsImm]==false) Format: **VertStride**
	115:113	**Src1.Width** Exists If: ([Src1.IsImm]==false) Format: **Width**
	112	**Src1.AddrMode** Exists If: ([Src1.IsImm]==false) Format: **AddrMode**
	111:98	**Src1.Operand** Exists If: ([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect) Format: **IndirectOperand**
	111:98	**Src1.Operand** Exists If: ([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct) Format: **DirectOperand**
	97:96	**Src1.HorzStride** Exists If: ([Src1.IsImm]==false) Format: **HorzStride**

mul - Multiply

	95:92	**CondCtrl**		
			Format:	**FlagModifier**
	91:88	**Src1.DataType**		
			Exists If:	([Src1.IsImm]==true)
			Format:	**ImmDataType**
	91:88	**Src1.DataType**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**RegDataType**
	87:84	**Src0.VertStride**		
			Format:	**VertStride**
	83:81	**Src0.Width**		
			Format:	**Width**
	80	**Src0.AddrMode**		
			Format:	**AddrMode**
	79:66	**Src0.Operand**		
			Exists If:	([Src0.AddrMode]==Direct)
			Format:	**DirectOperand**
	79:66	**Src0.Operand**		
			Exists If:	([Src0.AddrMode]==Indirect)
			Format:	**IndirectOperand**
	65:64	**Src0.HorzStride**		
			Format:	**HorzStride**
	63:50	**Dst.Operand**		
			Exists If:	([Dst.AddrMode]==Direct)
			Format:	**DirectOperand**
	63:50	**Dst.Operand**		
			Exists If:	([Dst.AddrMode]==Indirect)
			Format:	**IndirectOperand**
	49:48	**Dst.HorzStride**		
			Format:	**HorzStride**
	47	**Src1.IsImm** This field indicate that Source 1 operand is carrying an immediate value.		
			Value	**Name**
			0	false **[Default]**
			1	true

<table>
<tr><th colspan="3">mul - Multiply</th></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b>
<table><tr><td>Format:</td><td>SrcMod</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b>
<table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b>
<table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td>ImmDataType</td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b>
<table><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b>
<table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b>
<table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b>
<table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b>
<table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b>
<table><tr><td>Format:</td><td>MBZ</td></tr></table>
Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations</td></tr>
</table>

<table>
<tr><th colspan="3">mul - Multiply</th></tr>
<tr><td></td><td></td><td>supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td><b>PredCtrl</b></td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table>
<tr><td>Format:</td><td><b>ChanOff</b></td></tr>
</table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table>
<tr><td>Format:</td><td><b>ExecSize</b></td></tr>
</table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
</table>

mul - Multiply		
	15:0	**Header** Format: **Header**

Multiply Accumulate

mac - Multiply Accumulate

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

The mac instruction takes component-wise multiplication of src0 and src1, adds the results with the corresponding accumulator values, and then stores the final results in dst.

Format:

```
        [(pred)] mac[.cmod] (exec_size) dst src0 src1
```

Programming Notes

When source and destination datatypes are different, the implied datatype for the accumulator operand is always the destination datatype.

Integer source operands cannot be explicit accumulators.

Restriction

The conditional modifier and saturation (.sat) must not be used when src0 or src1 are dwords.

Syntax

```
[(pred)] mac[.cmod] (exec_size) reg reg reg
 [(pred)] mac[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = src0.chan[n] * src1.chan[n] + acc0.chan[n];
     }
 }
```

Src Types	Dst Types
*B,*W	*B,*W,*D
F	F
HF	HF
BF, F	BF, F

DWord	Bit	Description

mac - Multiply Accumulate

0..3	127:126	**Reserved**	
		Exists If:	([Src1.IsImm]==false)
		Format:	MBZ
	127:96	**Src1.ImmValue[31:0]**	
		Exists If:	([Src1.IsImm]==true)
	125:122	**Reserved**	
		Exists If:	([Src1.IsImm]==false)
		Format:	MBZ
	121:120	**Src1.Mod**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**SrcMod**
	119:116	**Src1.VertStride**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**VertStride**
	115:113	**Src1.Width**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**Width**
	112	**Src1.AddrMode**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**AddrMode**
	111:98	**Src1.Operand**	
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
		Format:	**IndirectOperand**
	111:98	**Src1.Operand**	
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
		Format:	**DirectOperand**
	97:96	**Src1.HorzStride**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**HorzStride**
	95:92	**CondCtrl**	
		Format:	**FlagModifier**
	91:88	**Src1.DataType**	
		Exists If:	([Src1.IsImm]==true)
		Format:	**ImmDataType**

mac - Multiply Accumulate

	91:88	**Src1.DataType** Exists If: ([Src1.IsImm]==false) Format: **RegDataType**
	87:84	**Src0.VertStride** Format: **VertStride**
	83:81	**Src0.Width** Format: **Width**
	80	**Src0.AddrMode** Format: **AddrMode**
	79:66	**Src0.Operand** Exists If: ([Src0.AddrMode]==Direct) Format: **DirectOperand**
	79:66	**Src0.Operand** Exists If: ([Src0.AddrMode]==Indirect) Format: **IndirectOperand**
	65:64	**Src0.HorzStride** Format: **HorzStride**
	63:50	**Dst.Operand** Exists If: ([Dst.AddrMode]==Direct) Format: **DirectOperand**
	63:50	**Dst.Operand** Exists If: ([Dst.AddrMode]==Indirect) Format: **IndirectOperand**
	49:48	**Dst.HorzStride** Format: **HorzStride**
	47	**Src1.IsImm** This field indicate that Source 1 operand is carrying an immediate value. Value | Name 0 | false **[Default]** 1 | true
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value. Value | Name 0 | false **[Default]** 1 | true

<table>
<tr><th colspan="3">mac - Multiply Accumulate</th></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td><b>Saturate</b></td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td><b>AccWrCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr></table></td></tr>
</table>

mac - Multiply Accumulate

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td>PredCtrl</td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td>ChanOff</td></tr></table>This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td>ExecSize</td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td>Header</td></tr></table></td></tr>
</table>

Multiply Accumulate High

mach - Multiply Accumulate High

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	true

The mach instruction performs DWord integer multiply-accumulate operation and outputs the high DWord (bits 63:32). For each enabled channel, this instruction multiplies the DWord in src0 with the high word of the DWord in src1, left shifts the result by 16 bits, adds it with the corresponding accumulator values, and keeps the whole 64-bit result in the accumulator. It then stores the high DWord (bits 63:32) of the results in dst.
This instruction is intended to be used to emulate 32-bit DWord integer multiplication by using the large number of bits available in the accumulator. Usage of accumulator content is restricted to the emulation sequence.
For example, the following instructions perform vector multiplication of two 32-bit signed integer sources from r2 and r3 and store the resulting vectors with the high 32 bits in r5 and the low 32 bits in r6.

```
mul (8) acc0:d r2.0<8;8,1>:d r3.0<16;8,2>:uw
 mach (8) r5.0<1>:d r2.0<8;8,1>:d r3.0<8;8,1>:d
 mov (8) r6.0<1>:d acc0:d // Low 32 bits.
```

Here is a different example including negation. An added preliminary mov is required for source modification on src1.

```
mov (8) r3.0<1>:d -r3<8;8,1>:d
 mul (8) acc0:d r2.0<8;8,1>:d r3.0<16;8,2>:uw
 mach (8) r5.0<1>:d r2.0<8;8,1>:d r3.0<8;8,1>:d // High 32 bits
 mov (8) r6.0<1>:d acc0:d // Low 32 bits.
```

The mach should have channel enable from the destHI of IMUL, the mov should have the channel enable from the destLO of IMUL. As mach is used to generate part of the 64-bit DWord integer results, saturation modifier should not be used. In fact, saturation modifier should not be used for any of these four instructions. Source and destination operands must be DWord integers. Source and destination must be of the same type, signed integer or unsigned integer. If dst is UD, src0 and src1 may be UD and/or D. However, if any of src0 and src1 is D, source modifier (abs) must be present to convert it to match with dst. If dst is D, src0 and src1 must also be D. They cannot be UD as it may cause unexpected overflow because the computed results are limited to 64 bits.

Format:

```
         [(pred)] mach[.cmod] (exec_size) dst src0 src1
```

Restriction
Accumulator is an implicit source and thus cannot be an explicit source operand.
The accumulator is an implicit destination and thus cannot be an explicit destination operand.

Syntax

```
[(pred)] mach[.cmod] (exec_size) reg reg reg
 [(pred)] mach[.cmod] (exec_size) reg reg imm32
```

mach - Multiply Accumulate High

Pseudocode

```
Evaluate(WrEn);
    for ( n = 0; n < exec_size; n++ ) {
       if ( WrEn.chan[n] ) {
          temp.chan[n][63:0] = (src1.chan[n][31:16] *
              src0.chan[n][31:0]) « 16 + acc.chan[n][63:0];
          if (AccWrEn) {
             acc.chan[n][63:0] = temp.chan[n][63:0];
             dst.chan[n][31:0] = temp.chan[n][63:32];
          }
          else {
             dst.chan[n][31:0] = temp.chan[n][31:0];
          }
        }
     }
```

Errata	Description
	A source modifier must not be used on src1 for the macro-operation. This applies to both mul and mach of the macro. If source modifier is required, an additional mov instruction may be used before the macro.

Src Types	Dst Types
D	D
UD	UD

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)
	125:122	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	121:120	**Src1.Mod** Exists If: ([Src1.IsImm]==false) Format: SrcMod
	119:116	**Src1.VertStride** Exists If: ([Src1.IsImm]==false) Format: VertStride
	115:113	**Src1.Width** Exists If: ([Src1.IsImm]==false) Format: Width

mach - Multiply Accumulate High

112	**Src1.AddrMode**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**AddrMode**
111:98	**Src1.Operand**		
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
		Format:	**IndirectOperand**
111:98	**Src1.Operand**		
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
		Format:	**DirectOperand**
97:96	**Src1.HorzStride**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**HorzStride**
95:92	**CondCtrl**		
		Format:	**FlagModifier**
91:88	**Src1.DataType**		
		Exists If:	([Src1.IsImm]==true)
		Format:	**ImmDataType**
91:88	**Src1.DataType**		
		Exists If:	([Src1.IsImm]==false)
		Format:	**RegDataType**
87:84	**Src0.VertStride**		
		Format:	**VertStride**
83:81	**Src0.Width**		
		Format:	**Width**
80	**Src0.AddrMode**		
		Format:	**AddrMode**
79:66	**Src0.Operand**		
		Exists If:	([Src0.AddrMode]==Direct)
		Format:	**DirectOperand**
79:66	**Src0.Operand**		
		Exists If:	([Src0.AddrMode]==Indirect)
		Format:	**IndirectOperand**
65:64	**Src0.HorzStride**		
		Format:	**HorzStride**

mach - Multiply Accumulate High

63:50 **Dst.Operand**

Exists If:	([Dst.AddrMode]==Direct)
Format:	**DirectOperand**

63:50 **Dst.Operand**

Exists If:	([Dst.AddrMode]==Indirect)
Format:	**IndirectOperand**

49:48 **Dst.HorzStride**

Format:	**HorzStride**

47 **Src1.IsImm**

This field indicate that Source 1 operand is carrying an immediate value.

Value	Name
0	false **[Default]**
1	true

46 **Src0.IsImm**

This field indicate that Source 0 operand is carrying an immediate value.

Value	Name
0	false **[Default]**
1	true

45:44 **Src0.Mod**

Format:	**SrcMod**

43:40 **Src0.DataType**

Exists If:	([Src0.IsImm]==false)
Format:	**RegDataType**

43:40 **Src0.DataType**

Exists If:	([Src0.IsImm]==true)
Format:	**ImmDataType**

39:36 **Dst.DataType**

Format:	**RegDataType**

35 **Dst.AddrMode**

Format:	**AddrMode**

34 **Saturate**

Format:	**Saturate**

33 **AccWrCtrl**

Format:	**AccWrCtrl**

32 **AtomicCtrl**

Format:	**AtomicCtrl**

<table>
<tr><th colspan="3">mach - Multiply Accumulate High</th></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr>
<tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b>
<table>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td>PredCtrl</td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
</table>

mach - Multiply Accumulate High

	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: **ChanOff** This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Multiply Add

mad - Multiply Add

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

Description

The mad instruction takes component-wise multiplication of src1 and src2, adds the results with the corresponding src0 values, and then stores the final results in dst.
The conditional modifier and saturation (.sat) must not be used when src1 or src2 are dwords.

```
Plane and Linear Interpolation instructions are removed. The following macros must be used
to emulate Plane and Linear Interpolation operations. Plane Instruction Emulation The
below plane instruction pln (16) r20.0<1>:f r10.4<0;1,0>:f r4.0<8;8,1>:fis emulated as
below mad (8) acc0<1>:f r10.7<0;1,0>:f r4.0<8;8,1>:f r10.4<0;1,0>:fmad (8) r20.0<1>:f
acc0<8;8,1>:f r5.0<8;8,1>:f r10.5<0;1,0>:fmad (8) acc0<1>:f r10.7<0;1,0>:f r6.0<8;8,1>:f
r10.4<0;1,0>:fmad (8) r21.0<1>:f acc0<8;8,1>:f r7.0<8;8,1>:f r10.5<0;1,0>:fIn case of
SIMD8 pln instruction only the first pair of mad instructions are used. Linear
Interpolation Instruction Emulation The below lrp instruction lrp (16) r40.0<1>:f
r10.0<8;8,1>:f r20.0<8;8,1>:f r30.0<8;8,1>:fis emulated as below mad (8) acc0<1>:f
r30.0<8;8,1>:f r10.0<8;8,1>:f r20.0<8;8,1>:fmad (8) r40.0<1>:f acc0<8;8,1>:f -
r10.0<8;8,1>:f r30.0<8;8,1>:fmad (8) acc0<1>:f r31.0<8;8,1>:f r11.0<8;8,1>:f
r21.0<8;8,1>:fmad (8) r41.0<1>:f acc0<8;8,1>:f -r11.0<8;8,1>:f r31.0<8;8,1>:fIn case of
SIMD8 lrp instruction only the first pair of mad instructions are used.
```

Format:

```
        [(pred)] mad[.cmod] (exec_size) dst src0 src1 src2
```

Restriction

Src1/Src2 for Integer source operands cannot be accumulators. Src0 is allowed to use accumulator.

When multiplying a DW and any lower precision integer, source modifier is not supported.

All three-source instructions have certain restrictions, described in Instruction Formats.

Syntax

```
[(pred)] mad[.cmod] (exec_size) reg  reg   reg reg
 [(pred)] mad[.cmod] (exec_size) reg  reg   reg imm16
 [(pred)] mad[.cmod] (exec_size) reg  imm16 reg reg
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = src1.chan[n] * src2.chan[n] + src0.chan[n];
     }
 }
```

mad - Multiply Add

Src Types	Dst Types
F	F
HF	HF
BF, F	BF, F
*B	*W
*W, *D	*W, *D

DWord	Bit	Description
0..3	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) AND ([Header][Opcode]!=madm) Format: **DirectOperand**
	127:114	**Src2.Operand** Exists If: ([Src2.IsImm]==false) AND ([Header][Opcode]==madm) Format: **MacroOperand**
	127:112	**Src2.ImmValue[15:0]** Exists If: ([Src2.IsImm]==true)
	113:112	**Src2.HorzStride** Exists If: ([Src2.IsImm]==false) Format: **HorzStride**
	111:98	**Src1.Operand** Exists If: ([Header][Opcode]!=madm) Format: **DirectOperand**
	111:98	**Src1.Operand** Exists If: ([Header][Opcode]==madm) Format: **MacroOperand**
	97:96	**Src1.HorzStride** Format: **HorzStride**
	95:92	**CondCtrl** Format: **FlagModifier**
	91	**Src1.VertStride[1]** Format: **TernaryVertStride[1:1]**
	90:88	**Src1.DataType** Format: **TernaryDataType**
	87:86	**Src1.Mod** Format: **SrcMod**
	85:84	**Src2.Mod** Format: **SrcMod**

mad - Multiply Add

83	**Src1.VertStride[0]**	
	Format:	**TernaryVertStride[0:0]**
82:80	**Src2.DataType**	
	Format:	**TernaryDataType**
79:66	**Src0.Operand**	
	Exists If:	([Src0.IsImm]==false) AND ([Header][Opcode]!=madm)
	Format:	**DirectOperand**
79:66	**Src0.Operand**	
	Exists If:	([Src0.IsImm]==false) AND ([Header][Opcode]==madm)
	Format:	**MacroOperand**
79:64	**Src0.ImmValue[15:0]**	
	Exists If:	([Src0.IsImm]==true)
65:64	**Src0.HorzStride**	
	Exists If:	([Src0.IsImm]==false)
	Format:	**HorzStride**
63:50	**Dst.Operand**	
	Exists If:	([Header][Opcode]!=madm)
	Format:	**DirectOperand**
63:50	**Dst.Operand**	
	Exists If:	([Header][Opcode]==madm)
	Format:	**MacroOperand**
49	**Reserved**	
	Format:	MBZ
48	**Dst.HorzStride** This field provides the distance in unit of data elements between two adjacent data elements within a row (horizontal) in the register region for the operand.	
	Value	**Name**
	0	1 element
	1	2 element
47	**Src2.IsImm** This field indicate that Source 2 operand is carrying an immediate value.	
	Value	**Name**
	0	false
	1	true

mad - Multiply Add

<table>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td>SrcMod</td></tr></table></td></tr>
<tr><td></td><td>43</td><td><b>Src0.VertStride[1]</b><table><tr><td>Format:</td><td>TernaryVertStride[1:1]</td></tr></table></td></tr>
<tr><td></td><td>42:40</td><td><b>Src0.DataType</b><table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
<tr><td></td><td>39</td><td><b>ExecDataType</b>
This field indicate the datatype mode of ternary instruction. Integer or Float.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Integer</td></tr><tr><td>1</td><td>Float</td></tr></table></td></tr>
<tr><td></td><td>38:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>TernaryDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Src0.VertStride[0]</b><table><tr><td>Format:</td><td>TernaryVertStride[0:0]</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
</table>

mad - Multiply Add

<table>
<tr><td></td><td>29</td><td><b>CmptCtrl</b>
<table><tr><td>Format:</td><td>MBZ</td></tr></table>
Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table><tr><td>Format:</td><td>PredCtrl</td></tr></table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table><tr><td>Format:</td><td>ChanOff</td></tr></table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
</table>

mad - Multiply Add

	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

No Operation

nop - No Operation

Source:	Eulsa
Length Bias:	4
Predication:	false
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

Do nothing. The nop instruction takes an instruction dispatch but performs no operation. It can be used for assembly patching in memory, or to insert a delay in the program sequence.

Format:

```
        nop
```

Restriction

The nop instruction takes no instruction options other than Breakpoint.

Syntax

```
nop
```

Pseudocode

```
{
     ;  // The null statement, which does nothing.
 }
```

DWord	Bit	Description
0..3	127:31	**Reserved** Access: RO Format: MBZ
	30	**Reserved**
	27:26	**Reserved** Format: MBZ
	25:18	**Reserved** Access: RO Format: MBZ
	17:16	**Reserved** Format: MBZ
	15:0	**Header** Format: **Header**

Oword Aligned Block Read MSD

MSD0R_OWAB - Oword Aligned Block Read MSD

Source:	EuSubFunctionDataPort0
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 01h Format: Opcode Aligned Block Read message
	13	**Block Message Subtype** Default Value: 0 Format: Opcode Oword Block Read/Write subtype
	12:11	**Reserved** Access: RO Format: MBZ
	10:8	**Data Elements** Format: MDC_DB_OW Specifies the number of contiguous Owords to be read

MSD0R_OWAB - Oword Aligned Block Read MSD

	7:0	**Binding Table Index** Format: **MDC_BTS_SLM_A32** Specifies the Binding Table Index for the message

Oword Aligned Block Write MSD

MSD0W_OWAB - Oword Aligned Block Write MSD

Source:	EuSubFunctionDataPort0
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 09h Format: Opcode Aligned Block Write message
	13	**Block Message Subtype** Default Value: 0 Format: Opcode Oword Block Read/Write subtype
	12:11	**Reserved** Access: RO Format: MBZ
	10:8	**Data Elements** Format: MDC_DB_OW Specifies the number of contiguous Owords to be written

MSD0W_OWAB - Oword Aligned Block Write MSD			
	7:0	**Binding Table Index**	
		Format:	MDC_BTS_SLM_A32
		Specifies the Binding Table Index for the message	

Oword Block Read MSD

MSD0R_OWB - Oword Block Read MSD

Source:	EuSubFunctionDataPort0
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 00h Format: Opcode Block Read message
	13	**Block Message Subtype** Default Value: 0 Format: Opcode Oword Block Read/Write subtype
	12:11	**Reserved** Access: RO Format: MBZ
	10:8	**Data Elements** Format: MDC_DB_OW Specifies the number of contiguous Owords to be read or written

MSD0R_OWB - Oword Block Read MSD		
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Oword Block Write MSD

MSD0W_OWB - Oword Block Write MSD		
Source:	EuSubFunctionDataPort0	
Length Bias:	1	
DWord	**Bit**	**Description**
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 08h Format: Opcode Block Write message
	13	**Block Message Subtype** Default Value: 0 Format: Opcode Oword Block subtype
	12:11	**Reserved** Access: RO Format: MBZ
	10:8	**Data Elements** Format: MDC_DB_OW Specifies the number of contiguous Owords to be read or written

MSD0W_OWB - Oword Block Write MSD		
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

PIPE_CONTROL

PIPE_CONTROL

Source:	RenderCS, ComputeCS
Length Bias:	2

The PIPE_CONTROL command is used to effect the synchronization described above.

Programming Notes	Source
SW must follow below programming restrictions when programming PIPE_CONTROL command: • "Command Streamer Stall Enable" must be always set. • Post Sync Operations must not be set to Write PS Depth Count • Following bits must not be set when programmed for ComputeCS • "Render Target Cache Flush Enable", "Depth Cache Flush Enable" and "Tile Cache Flush Enable" • "Depth Stall Enable", Stall at Pixel Scoreboard and "PSD Sync Enable". • "OVR Tile 0 Flush", "TBIMR Force Batch Closure", "AMFS Flush Enable" "VF Cache Invalidation Enable" and "Global Snapshot Count Reset".	ComputeCS

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 2h PIPE_CONTROL Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0h PIPE_CONTROL Format: OpCode
	15	**Reserved** Access: RO Format: MBZ
	14	**Workload Partition ID Offset Enable** **Description** This bit controls the memory write address computation for the store data update. The final memory write address is computed by adding the Workload Partition ID times the Address Offset to the memory address mentioned in the command. Workload Partition ID gets

PIPE_CONTROL

programmed through WPARID register and the Address Offset gets programmed through CS_MI_ADDRESS_OFFSET register.
Example: Final Memory Write Address[47:2] = (Workload Partition ID* "Address Offset") + Memory Write Address [47:2]

Value	Name	Description
1		The final memory address is computed based on the Workload Partition ID
0		There is no offset added to the memory write address.

13 **Compression Control Surface (CCS) Flush**

This bit controls the flushing of the engine (Render, Compute) specific entries from the compression cache.

12 **Reserved**

Access:	RO
Format:	MBZ

11 **Untyped Data-Port Cache Flush**

This bit controls the flushing of the data-port's Untyped L1 data cache (LSC L1).
If set, dataport ensures all the dirty lines are evicted, and clean lines are invalidated, in the subslice Untyped L1 data cache.

Programming Notes	Source
• "HDC Pipeline Flush" bit must be set for this bit to take effect.	
This bit is functional only when PIPELINE_SELECT command has set “Pipeline Select” mode to “GPGPU”. This bit is ignored when PIPELINE_SELECT is not "GPGPU". When the "Pipeline Select" mode is set to "3D", the LSC L1 cache flush/invalidate is controlled by the "HDC Pipeline Flush" field in this command.	RenderCS

10 **L3 Read Only Cache Invalidation Enable**

Description
This bit controls the invalidation of the L3 Read Only Cache at the top of the pipe, i.e. at command parsing time. Setting this bit is independent of any other bit in this packet.
This bit controls the invalidation of the Geometry streams cached in L3 Cache at the top of the pipe, i.e. at command parsing time. Setting this bit is independent of any other bit in this packet.

9 **HDC Pipeline Flush**

If set, HDC and LSC ensures it's pipeline is flushed and the memory transactions are "globally observed" to its coherency point as part of the flush operation. The HDC read-only cache is also flushed as well. This will not result in flushing L3$ portion that caches dataport writes.

PIPE_CONTROL

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Programming Notes</th><th>Source</th></tr>
<tr><td>When the "Pipeline Select" mode in <b>PIPELINE_SELECT</b> command is set to "3D", HDC Pipeline Flush can also flush/invalidate the LSC Untyped L1 cache.
When the "Pipeline Select" mode is set to "GPGPU", the LSC Untyped L1 cache flush is controlled by "Untyped Data-Port Cache Flush" bit in the PIPE_CONTROL command.</td><td>RenderCS</td></tr>
</table>
</td></tr>
<tr><td></td><td>8</td><td><b>Predicate Enable</b>
If set, this command is executed (or not) depending on the current value of the MI Predicate internal state bit (MI_PREDICATE_RESULT). This command is ignored (NOOP'd) if PredicateEnable is set and the Predicate state (MI_PREDICATE_RESULT[0])) bit is 0.
This command is un-conditionally NOOP'd when MI_SET_PREDICATE_RESULT[0] is set.</td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Default Value:</td><td>4h DWORD_COUNT_n</td></tr>
<tr><td>Format:</td><td>=n</td></tr>
</table>
Total Length - 2. Excludes DWord (0,1).</td></tr>
<tr><td>1</td><td>31</td><td><b>TBIMR Force Batch Closure</b>
This bit forces SF to close the batch. This bit must be set with a post sync operation.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>No Batch Closure</td><td>SF will not close the batch on receiving marker.</td></tr>
<tr><td>1</td><td>Close Batch</td><td>SF will close the batch on receiving the marker associated with this command.</td></tr>
</table>
</td></tr>
<tr><td></td><td>30</td><td><b>L3 Fabric Flush</b>
<table>
<tr><th>Description</th></tr>
<tr><td>L3 Fabric Flush will ensure all the pending transactions in the L3 Fabric are flushed to global observation point. HW does implicit L3 Fabric Flush on all stalling flushes (both explicit and implicit) and on PIPECONTROL having Post Sync Operation enabled. This bit provides an explicit control.</td></tr>
<tr><td><b>Pipelined L3 Fabric Flush:</b>
When Depth Stall Enable is set L3 Fabric Flush is pipelined along with the workload and issued from raster stage in the pipeline. Flush marker for this flush type on reaching raster stage, will ensure all the prior workload is complete and then L3 Fabric Flush is performed before allowing the workload following the flush marker to execute.
By default L3 Fabric Flush happens top of the pipe as part of post-synchronization operation on a flush completion.
<b>Usages:</b>
This mechanism is used to flush the updated compressed control state of an surface during fast clear to global observable point before the rendering operations are started using these surfaces. Most common usage case is to flush the L1 (Color, Depth) caches with L3 Fabric Flush and Depth Stall Enable post Fast Clears prior to starting the rendering operations. Refer Render Target Fast Clear and Depth Buffer Clear sections for more details.
- For a sequence of color fast clears</td></tr>
</table>
</td></tr>
</table>

PIPE_CONTROL

- A single PIPE_CONTROL command with Render Target Cache Flush, L3 Fabric Flush and Depth Stall set at the end of the sequence suffices. This assumes the output of the fast clears are used only post this PIPE_CONTROL command.

- For a sequence of depth fast clears

- A single PIPE_CONTROL command with Depth Cache Flush, L3 Fabric Flush and Depth Stall set at the end of the sequence suffices. This assumes the output of the depth fast clears are used only post this PIPE_CONTROL command.

- For a sequence of mixed color/depth fast clears.

- A single PIPE_CONTROL command with Depth Cache Flush, Render Target Cache Flush, L3 Fabric Flush and Depth Stall set at the end of the sequence suffices. This assumes the output of the color/depth fast clears are used only post this PIPE_CONTROL command.

Value	Name	Description
1		HW will do a L3 Fabric Flush on completion of flush for the corresponding PIPECONTROL. Setting this bit will force HW to send a marker downstream for a flush completion.
0		HW will not force a "L3 Fabric Flush" on completion of flush and will only do on need basis.

29 **Command Cache Invalidate Enable**

Format:	Enable

When set the command cache for commands parsed at the top of the pipe will be invalidated. This bit is independent from the other bits in this command and will be executed prior to the pipeline being flushed.

28 **Tile Cache Flush Enable**

Setting this bit will force Tile Cache (contains both color and depth data) to be flushed to memory prior to this synchronization point completing. This bit must be set for all write fence sync operations to assure that results from operations initiated prior to this command are visible in memory once software observes this synchronization.

Value	Name	Description
0		Tile Cache is not flushed.
1		Tile cache is flushed.

Programming Notes

Tile Cache Enabled Mode:

- SW must always set CS Stall bit when Tile Cache Flush Enable bit is set in the PIPECONTROL command.
- SW must ensure level1 depth and color caches are flushed prior to flushing the tile cache. This can be achieved by following means:
  - Single PIPECONTROL command to flush level1 caches and the tile cache. Hardware will sequence the flushing of L1 caches followed by the Tile cache.

Attributes listed below must be set. OR

  - Tile Cache Flush Enable
  - Render Target Cache Flush Enable
  - Depth Cache Flush Enable

- Flushing of L1 caches followed by flushing of tile cache through two different PIEPCONTROL commands. SW must ensure not to issue any rendering commands between the two PIPECONTROL commands.

Unified Cache (Tile Cache Disabled):
In unified cache mode of operation Color and Depth (Z) streams are cached in DC space of L2 along with Data Port stream. On a Tile Cache Flush only Color and Depth (Z) streams from DC space of L2 are flushed to globally observable and whereas DC Flush Enable will only flush Data Port stream from the DC space of L2 to globally observable. Refer L3 configuration section for Unified cache usage model. In this mode of operation there is no dedicated memory allocated for Tile Cache in L2. When the Color and Depth (Z) streams are enabled to be cached in the DC space of L2, Software must use Render Target Cache Flush Enable and Depth Cache Flush Enable along with Tile Cache Flush for getting the color and depth (Z) write data to be globally observable. In this mode of operation it is not required to set CS Stall upon setting Tile Cache Flush bit.

27 **Reserved**

26 **Flush LLC**

Format:	Enable

If enabled, at the end of the current pipe-control the last level cache is cleared of all the cachelines which have been determined as being part of the Frame Buffer.

Programming Notes
SW must always program Post-Sync Operation to "Write Immediate Data" when Flush LLC is set.

25 **AMFS Flush Enable**

Format:	Enable

If enabled, at the end of the current pipe-control the AMFS unit stalls until all spawned texel shaders are completed, and then the AMFS unit flushes internal cache(s) to memory.
This bit should be enabled when a procedural texture transitions from the write state to the read state.

24 **Destination Address Type**

Defines address space of Destination Address

Value	Name	Description
0h	PPGTT	Use PPGTT address space for DW write
1h	GGTT	Use GGTT address space for DW write

Programming Notes
Ignored if ""No Write" is selected in Operation.

PIPE_CONTROL

<table>
<tr><td></td><td>23</td><td><b>LRI Post Sync Operation</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>No LRI Operation</td><td>No LRI operation occurs as a result of this instruction. The Post-Sync Operation field is valid and may be used to specify an operation.</td></tr>
<tr><td>1h</td><td>MMIO Write Immediate Data</td><td>Write the DWord contained in Immediate Data Low (DW3) to the MMIO offset specified in the Address field.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This bit causes a post sync operation with an LRI (Load Register Immediate) operation. If this bit is set then the Post-Sync Operation field must be cleared.</td></tr>
</table>
</td></tr>
<tr><td></td><td>22</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>21</td><td><b>Store Data Index</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is index into the global hardware status page when destination address type in the command is set to 1 (GGTT). The store data address is index into the per-process hardware status page when destination address type in the command is set to 0 (PPGTT).</td></tr>
<tr><td></td><td>20</td><td><b>Command Streamer Stall Enable</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
If ENABLED, the sync operation will not occur until all previous flush operations pending a completion of those previous flushes will complete, including the flush produced from this command. This enables the command to act similar to the legacy MI_FLUSH command.</td></tr>
<tr><td></td><td>19</td><td><b>Depth Stall Sync Enable</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
If set, 3D pipeline will stall any subsequent primitives at the Depth Test stage until they Sync across all the slices. Once all the Depth Test Stages are synced up (across Slices), post-sync operations take place and then they get uninstalled.</td></tr>
<tr><td></td><td>18</td><td><b>TLB Invalidate</b>
<table>
<tr><td>Format:</td><td>U1</td></tr>
</table>
<table>
<tr><td>If ENABLED, all TLBs belonging to Render Engine will be invalidated once the flush operation is complete. Note that if the flush TLB invalidation mode is clear, a TLB invalidate will occur irrespective of this bit setting</td></tr>
<tr><td>If ENABLED, PIPE_CONTROL command will flush the in flight data written out by render engine to Global Observation point on flush done. Also Requires stall bit ([20] of DW1) set.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>If ENABLED, all TLBs belonging to Render Engine will be invalidated once the flush operation is complete. Note that if the flush TLB invalidation mode is clear, a TLB invalidate will occur irrespective of this bit setting.</td></tr>
</table>
</td></tr>
</table>

PIPE_CONTROL

Post Sync Operation or CS stall must be set to ensure a TLB invalidate occurs. Otherwise no cycle will occur to the TLB cache to invalidate.

17 **PSS Stall Sync Enable**

Format:	Enable

If set, PSS Units will stall successive PS threads from being dispatched until all the prior PS threads complete. Once all PSSs are synced up (across Slices), post-sync operations take place and then PSS units will get uninstalled.

16 **Generic Media State Clear**

Format:	Disable

If set, all generic media state context information will be invalidated. Any state invalidated will not be saved as part of the render engine context image. The state only becomes valid once it is parsed by the command streamer.

Programming Notes
Ignored. Not needed with COMPUTE_WALKER command.

15:14 **Post Sync Operation**

Description
This field specifies an optional action to be taken upon completion of the synchronization operation.
This field must be cleared if the LRI Post-Sync Operation bit is set.

Value	Name	Description	Programming Notes
0h	No Write	No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc.	
1h	Write Immediate Data	Write the QWord containing Immediate Data Low, High DWs to the Destination Address	
2h	Write PS Depth Count	Write the 64-bit PS_DEPTH_COUNT register to the Destination Address	Workaround : Driver must program PIPE_CONTROL with only Depth Stall Enable bit set prior to programming a PIPE_CONTROL with Write PS Depth Count Post sync operation.
3h	Write Timestamp	Write the 64-bit TIMESTAMP register(i.e. "Reported Timestamp Count" 0x2358 for render pipe) to the Destination Address.	

Programming Notes
If executed in non-secure batch buffer, the address given will be in a PPGTT address space. If in a secure ring or batch, address given will be in GGTT space

PIPE_CONTROL

13 **Depth Stall Enable**

Format:	Enable

This bit must be set when obtaining a "visible pixel" count to preclude the possible inclusion in the PS_DEPTH_COUNT value written to memory of some fraction of pixels from objects initiated after the PIPE_CONTROL command.

Value	Name	Description
0h	Disable	3D pipeline will not stall subsequent primitives at the Depth Test stage.
1h	Enable	3D pipeline will stall any subsequent primitives at the Depth Test stage until the Sync and Post-Sync operations complete.

Programming Notes
This bit must be DISABLED for operations other than writing PS_DEPTH_COUNT.
This bit will have no effect (besides preventing write cache flush) if set in a PIPE_CONTROL command issued to the Media pipe.

12 **Render Target Cache Flush Enable**

Format:	Enable

Setting this bit will force Render Cache to be flushed to memory prior to this synchronization point completing. This bit must be set for all write fence sync operations to assure that results from operations initiated prior to this command are visible in memory once software observes this synchronization.

Value	Name	Description
0h	Disable Flush	Render Target Cache is NOT flushed.
1h	Enable Flush	Render Target Cache is flushed.

Programming Notes
Whenever a Binding Table Index (BTI) used by a Render Target Message points to a different RENDER_SURFACE_STATE, SW must issue a Render Target Cache Flush by enabling this bit.
When render target flush is set due to new association of BTI, PS Scoreboard Stall bit must be set in this packet.

11 **Instruction Cache Invalidate Enable**

Format:	Enable

Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the L1 and L2 at the top of the pipe i.e. at the parsing time.

10 **Texture Cache Invalidation Enable**

Format:	Enable

Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the texture caches at the top of the pipe i.e. at the parsing time.

9 **Indirect State Pointers Disable**

Format:	Enable

<table>
<tr><th colspan="3">PIPE_CONTROL</th></tr>
<tr><td></td><td></td><td><table><tr><th>Description</th></tr><tr><td>At the completion of the post-sync operation associated with this pipe control packet, the indirect state pointers in the hardware are considered invalid; the indirect pointers are not saved in the context. If any new indirect state commands are executed in the command stream while the pipe control is pending, the new indirect state commands are preserved.</td></tr><tr><td>Using Invalidate State Pointer (ISP) only inhibits context restoring of Push Constant (3DSTATE_CONSTANT_*) commands. Push Constant commands are only considered as Indirect State Pointers. Once ISP is issued in a context, SW must initialize by programming push constant commands for all the shaders (at least to zero length)before attempting any rendering operation for the same context.</td></tr></table></td></tr>
<tr><td></td><td>8</td><td><b>Notify Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in Memory Interface Registers for details.</td></tr>
<tr><td></td><td>7</td><td><b>Pipe Control Flush Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>Hardware on parsing PIPECONTROL command with Pipe Control Flush Enable set will wait for all the outstanding post sync operations corresponding to previously executed PIPECONTROL commands are complete before making forward progress.</td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>5</td><td><b>DC Flush Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>Setting this bit enables flushing of the L3$ portions that caches DC writes.<table><tr><th>Programming Notes</th></tr><tr><td>DC Flush (L3 Flush)by default doesn’t result in flushing/invalidating the IA Coherent lines from L3$, however this can be achieved by setting control bit <b>Pipe line flush Coherent lines</b> in L3SQCREG4 register.</td></tr></table></td></tr>
<tr><td></td><td>4</td><td><b>VF Cache Invalidation Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of VF address based cache at the top of the pipe i.e. at the parsing time.</td></tr>
<tr><td></td><td>3</td><td><b>Constant Cache Invalidation Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the constant cache at the top of the pipe i.e. at the parsing time.</td></tr>
<tr><td></td><td>2</td><td><b>State Cache Invalidation Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the L1 and L2 state caches at the top of the pipe i.e. at the parsing time.</td></tr>
</table>

<table>
<tr><th colspan="3">PIPE_CONTROL</th></tr>
<tr><td></td><td>1</td><td><b>Stall At Pixel Scoreboard</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
Defines the behavior of PIPE_CONTROL command at the pixel scoreboard.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable</td><td>Stall at the pixel scoreboard is disabled.</td></tr>
<tr><td>1h</td><td>Enable</td><td>Stall at the pixel scoreboard is enabled.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This bit must be DISABLED for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries. This bit is ignored if Depth Stall Enable is set. Further the render cache is not flushed even if Write Cache Flush Enable bit is set.</td></tr>
<tr><td>Deprecated. Use <b>PSS Stall Sync Enable.</b></td></tr>
</table></td></tr>
<tr><td></td><td>0</td><td><b>Depth Cache Flush Enable</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
Setting this bit enables flushing (i.e. writing back the dirty lines to memory and invalidating the tags) of depth related caches. This bit applies to HiZ cache, Stencil cache and depth cache.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Flush Disabled</td><td>Depth relates caches (HiZ, Stencil and Depth) are NOT flushed.</td></tr>
<tr><td>1h</td><td>Flush Enabled</td><td>Depth relates caches (HiZ, Stencil and Depth) are flushed.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Ideally depth caches need to be flushed only when depth is required to be coherent in memory for later use as a texture, source or honoring CPU lock. This bit must be DISABLED for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries.</td></tr>
</table></td></tr>
<tr><td>2</td><td>31:2</td><td><b>Address</b>
<table><tr><td>Format:</td><td>GraphicsAddress[31:2]U32</td></tr></table>
If <b>Post Sync Operation</b> is set to 1h: <b>LRI Post-Sync Operation</b> must be clear): Bits 31:3 specify the QW address of where the Immediate Data following this DW in the packet to be stored. Bit 2 MBZ Ignored if "No Write" is the selected in Post-Sync Operation: If <b>LRI Post-Sync Operation</b> is set: Bits 22:2 (Bits 31:23 are reserved MBZ) specify the MMIO offset destination for the data in the <b>Immediate Data Low</b> (DW3) field. Only DW writes are valid.</td></tr>
<tr><td></td><td>1:0</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>3</td><td>31:0</td><td><b>Address High</b>
<table><tr><td>Format:</td><td>GraphicsAddress[63:32]U32</td></tr></table>
This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space. This field is valid only if the post-sync operation is not 0 and the LRI Post-Sync Operation is clear.</td></tr>
</table>

PIPE_CONTROL		
4..5	63:0	**Immediate Data** Format: | U64 This field specifies the QWord value to be written to the targeted location. Only valid when Post-Sync Operation is 1h (Write Immediate Data) or LRI Post-Sync Operation is set. Ignored if Post-Sync Operation is "No write", "Write PS_DEPTH_COUNT" or "Write TIMESTAMP".

PIPELINE_SELECT

PIPELINE_SELECT

Source:	RenderCS, ComputeCS
Length Bias:	1

Description
The PIPELINE_SELECT command is used to specify which GPE pipeline is to be considered the 'current' active pipeline.
Issuing 3D-pipeline-specific commands when the GPGPU pipeline is selected, or vice versa, is UNDEFINED.
Programming common non pipeline commands (e.g., STATE_BASE_ADDRESS) is allowed in all pipeline modes.

Programming Notes
Software must ensure Render Cache, Depth Cache and HDC Pipeline flush are flushed through a stalling PIPE_CONTROL command prior to programming of PIPELINE_SELECT command transitioning Pipeline Select from 3D to GPGPU/Media. Software must ensure HDC Pipeline flush and Generic Media State Clear is issued through a stalling PIPE_CONTROL command prior to programming of PIPELINE_SELECT command transitioning Pipeline Select from GPGPU/Media to 3D. Example: • Workload-3Dmode, • PIPE_CONTROL (CS Stall, Depth Cache Flush Enable, Render Target Cache Flush Enable, HDC Pipeline Flush Enable) , • PIPELINE_SELECT (GPGPU), • Workload-GPGPU mode, • PIPE_CONTROL (CS Stall, HDC Pipeline Flush Enable, Generic Media State Clear), • PIPELINE_SELECT (3D) ...
"Pipe Selection" must be never set to "3D" in PIPELINE_SELECT command programmed for workloads submitted to ComputeCS.
While GPU is operating in GPGPU mode of operation and when a Mid Thread Preemption (if enabled) occurs on a PIPELINE_SELECT command with Media Sampler DOP CG Enable reset along with Pipeline Select Mode set to 3D and on resubmission of this context on context restore Sampler DOP CG Enable will be reset. This would mean the GPGPU mid thread preempted threads restored will get executed with media sampler DOP clock not gated consuming media sampler DOP power until all GPGPU threads have retired. Programming of the PIPELINE_SELECT can be modified to avoid the above inefficiency. This can be done by programming Pipeline Selection and Media Sampler DOP CG Enable fields in two different PIPELINE_SELECT commands instead of on single PIPELINE_SELECT command. Example: PIPELINE_SELECT (Pipeline Selection = 3D, Media Sampler DOP CG Enable = False) To PIPELINE_SELECT (Pipeline Selection = 3D) PIPELINE_SELECT (Media Sampler DOP CG Enable = False)

DWord	Bit	Description

<table>
<tr><th colspan="3">PIPELINE_SELECT</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h GFXPIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Command SubType</b><table><tr><td>Default Value:</td><td>1h GFXPIPE_SINGLE_DW</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><b>3D Command Opcode</b><table><tr><td>Default Value:</td><td>1h GFXPIPE_NONPIPELINED</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>3D Command Sub Opcode</b><table><tr><td>Default Value:</td><td>04h PIPELINE_SELECT</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:8</td><td><b>Mask Bits</b><table><tr><th>Programming Notes</th></tr><tr><td>Must be set to modify corresponding bits in Bits 7:0. (For implemented bits)</td></tr></table></td></tr>
<tr><td></td><td>7</td><td><b>Systolic Mode Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>When set, this will enable systolic mode for the following COMPUTE_WALKER commands. This will lower the Fmax to avoid ICC current issues when executing systolic array commands in the execution units. If this is not set prior to executing systolic array operations, the context will be halted to avoid any ICC issues.<table><tr><th>Value</th><th>Name</th></tr><tr><td>1</td><td>Systolic Mode Enabled</td></tr><tr><td>0</td><td>Systolic Mode Disabled <b>[Default]</b></td></tr></table></td></tr>
<tr><td></td><td>6</td><td><b>Media Sampler Power Clock Gate Disable</b><table><tr><td>Format:</td><td>U1</td></tr></table>By default, the media power clock gating is always ON.
When set, Command Streamer sends message to PM to disable media sampler power Clock Gating.<table><tr><th>Programming Notes</th></tr><tr><td>Mask bit [14] has to be set for HW to look at this field when PIPELINE_SELECT command is parsed.
Each BB/workload is responsible to set this control. This can be only enabled/disabled at the frame level.</td></tr></table></td></tr>
<tr><td></td><td>5</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

PIPELINE_SELECT

4 **Reserved**

Access:	RO
Format:	MBZ

3 **Render Sampler Power Gate Enable**

Format:	Enable

Value	Name	Description
0	Disabled	Command Streamer sends message to PM to disable render sampler Power Gating.
1	Enabled	Command Streamer sends message to PM to enable render sampler Power Gating.

Programming Notes
Mask bit [11] has to be set for HW to look at this field when PIPELINE_SELECT command is parsed.

2 **Render Slice common Power Gate Enable**

Format:	Enable

Value	Name	Description
0	Disabled	Command Streamer sends message to PM to disable render slice common Power Gating.
1	Enabled	Command Streamer sends message to PM to enable render slice common Power Gating.

Programming Notes
Mask bit [10] has to be set for HW to look at this field when PIPELINE_SELECT command is parsed.

1:0 **Pipeline Selection**

Value	Name	Description
0	3D	3D pipeline is selected
2	GPGPU	GPGPU pipeline is selected

Programming Notes
Mask bits [9:8] has to be set for HW to look at this field when PIPELINE_SELECT command is parsed. Setting only one of the mask bit [9] or [8] is illegal.

Read State Information

DP_RSI - Read State Information

Source:	SFID_D
Length Bias:	1

Return state information based on the message bindless surface state offset, surface state offset or binding table index.

Programming Notes
Exec_mask is ignored for this message

Syntax
RSI.sfid dest_reg <addr_type+state_offset>src0_nullreg

Pseudocode
`dest_reg[0] = READ_SURFACE_STATE(state_offset)`

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>30:29</td><td><b>Address Type</b>
<table>
<tr><td>Format:</td><td>DP_ADDR_SURFACE_TYPE</td></tr>
</table>
Specifies the format of the Extended Descriptor used with the address payload.
<table>
<tr><th>Restriction</th></tr>
<tr><td>DP_ADDR_TYPE must be BSS, SS or BTI.</td></tr>
</table></td></tr>
<tr><td></td><td>28:25</td><td><b>Src0 Length</b>
<table>
<tr><td>Format:</td><td>DP_ONE_ADDR_REG</td></tr>
</table>
Specifies the size of the address payload (ASTATE_INFO_PAYLOAD), in registers.</td></tr>
<tr><td></td><td>24:20</td><td><b>Dest Length</b>
<table>
<tr><td>Format:</td><td>U5</td></tr>
</table>
Specifies the size of destination data register payload (STATE_INFO_PAYLOAD).
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>2</td><td></td><td>This message returns 2 registers.</td></tr>
</table></td></tr>
<tr><td></td><td>19:17</td><td><b>RSI Sub-opcode</b>
<table>
<tr><td>Format:</td><td>U3</td></tr>
</table>
Specifies the sub-opcode for state read message.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Programming Notes</th></tr>
<tr><td>0</td><td>Read Surface</td><td>Return 64B raw surface state data, using the BSS offset, SS offset or BTI from the message descriptor. Surface</td><td>Read Surface State is only supported for</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">DP_RSI - Read State Information</th></tr>
<tr><td></td><td></td><td><table><tr><td></td><td>State</td><td>state BSS and SS offsets are 64B aligned. The 26-bit BSS/SS offset from the message descriptor is mapped to surface_state_[B]SS_offset[31:6].</td><td>DP_ADDR_TYPE as BSS, SS, BTI.</td></tr></table></td></tr>
<tr><td></td><td>16:6</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>RSI</b><table><tr><td>Default Value:</td><td>30 RSI</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Read Surface Info MSD

MSD_RSI - Read Surface Info MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHF Indicates that the message forbids a header.
	18	**Reserved** Access: RO Format: MBZ
	17:14	**Message Type** Default Value: 06h Format: Opcode Read Surface Info message
	13:8	**Reserved** Access: RO Format: MBZ
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

REP16 Render Target Write MSD

MSD_RTW_REP16 - REP16 Render Target Write MSD

Source:	EuSubFunctionRenderDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message
	29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Per-Coarse Pixel PS outputs enable** Format: Enable This bit indicates the render target write is a coarse pixel write. **Programming Notes** This bit must be DISABLED if a pixel shader thread is dispatched at pixel- or sample- rate. This bit may be DISABLED if a multi-rate pixel shader thread is dispatched at coarse rate. In such case, it indicates per-pixel or per-sample write (as determined by Per-Sample PS outputs enable) from a multi-rate shader, where the set of pixels is indicated by the Pixel shading phase field in Extended Message descriptor.
	17:14	**Message Type** Default Value: 0Ch Format: Opcode Render Target Write message

MSD_RTW_REP16 - REP16 Render Target Write MSD

	13	**Per-Sample PS Enable** Format: Enable If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot. **Programming Notes** This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Last Render Target Select** Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.
	11	**Slot Group Select** Format: MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
	10:8	**Render Target Message Subtype** Default Value: 1h Format: Opcode SIMD16 Single source message with replicated data. Use slots [15:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Reserved Instruction0

Reserved Instruction0		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000003 Format: Opcode
	26:24	**Opcode 2** Default Value: 0x00000000 Format: Opcode
	23:16	**Opcode 3** Default Value: 0x00000053 Format: Opcode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction1

Reserved Instruction1

Length Bias: 1

DWord	Bit	Description
0	31:29	**Opcode 0** Default Value: 0x00000000 Format: Opcode
	28:23	**Opcode 1** Default Value: 0x0000000E Format: Opcode
	22:0	**Reserved** Access: RO Format: MBZ
0..n	31:0	**Unknown Bitfield**

Reserved Instruction2

Reserved Instruction2

Length Bias: 1

DWord	Bit	Description
0	31:29	**Opcode 0** Default Value: 0x00000000 Format: Opcode
	28:23	**Opcode 1** Default Value: 0x0000000E Format: Opcode
	22:0	**Reserved** Access: RO Format: MBZ
0..n	31:0	**Unknown Bitfield**

Reserved Instruction3

Reserved Instruction3		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:16	**Opcode 1** Default Value: 0x00001608 Format: Opcode
	15:0	**Reserved** Access: RO Format: MBZ
0..n	31:0	**Unknown Bitfield**

Reserved Instruction4

Reserved Instruction4		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:16	**Opcode 1** Default Value: 0x00001600 Format: Opcode
	15:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction5

Reserved Instruction5		
Length Bias: 2		
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:16	**Opcode 1** Default Value: 0x0000160A Format: Opcode
	15:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction6

Reserved Instruction6

Length Bias: 2

DWord	Bit	Description
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:16	**Opcode 1** Default Value: 0x00001609 Format: Opcode
	15:0	**Reserved** Access: RO Format: MBZ
0..n	31:0	**Unknown Bitfield**

Reserved Instruction7

Reserved Instruction7		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:24	**Opcode 2** Default Value: 0x00000000 Format: Opcode
	23:21	**Opcode 3** Default Value: 0x00000000 Format: Opcode
	20:16	**Opcode 4** Default Value: 0x00000005 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction8

Reserved Instruction8

Length Bias: 2

DWord	Bit	Description	
0	31:29	**Opcode 0**	
		Default Value:	0x00000003
		Format:	Opcode
	28:27	**Opcode 1**	
		Default Value:	0x00000002
		Format:	Opcode
	26:24	**Opcode 2**	
		Default Value:	0x00000001
		Format:	Opcode
	23:21	**Opcode 3**	
		Default Value:	0x00000000
		Format:	Opcode
	20:16	**Opcode 4**	
		Default Value:	0x00000000
		Format:	Opcode
	15:12	**Reserved**	
		Access:	RO
		Format:	MBZ
	11:0	**DWord Count**	
		Format:	=n
0..n	31:0	**Unknown Bitfield**	

Reserved Instruction9

Reserved Instruction9		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x00000003 Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000007 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction10

Reserved Instruction10

Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x00000003 Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000006 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction11

Reserved Instruction11				
Length Bias:	2			
DWord	**Bit**	**Description**		
0	31:29	**Opcode 0**		
			Default Value:	0x00000003
			Format:	Opcode
	28:27	**Opcode 1**		
			Default Value:	0x00000002
			Format:	Opcode
	26:23	**Opcode 2**		
			Default Value:	0x00000007
			Format:	Opcode
	22:16	**Opcode 3**		
			Default Value:	0x00000007
			Format:	Opcode
	15:12	**Reserved**		
			Access:	RO
			Format:	MBZ
	11:0	**DWord Count**		
			Format:	=n
0..n	31:0	**Unknown Bitfield**		

Reserved Instruction12

Reserved Instruction12

Length Bias: 2

DWord	Bit	Description
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x00000007 Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000006 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction13

Reserved Instruction13		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x00000007 Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000022 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction14

Reserved Instruction14		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x00000007 Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000021 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction15

<table>
<tr><th colspan="3">Reserved Instruction15</th></tr>
<tr><td>Length Bias:</td><td colspan="2">2</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Opcode 0</b>
Default Value: 0x00000003
Format: Opcode</td></tr>
<tr><td></td><td>28:27</td><td><b>Opcode 1</b>
Default Value: 0x00000002
Format: Opcode</td></tr>
<tr><td></td><td>26:23</td><td><b>Opcode 2</b>
Default Value: 0x00000007
Format: Opcode</td></tr>
<tr><td></td><td>22:16</td><td><b>Opcode 3</b>
Default Value: 0x00000010
Format: Opcode</td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b>
Access: RO
Format: MBZ</td></tr>
<tr><td></td><td>11:0</td><td><b>DWord Count</b>
Format: =n</td></tr>
<tr><td>0..n</td><td>31:0</td><td><b>Unknown Bitfield</b></td></tr>
</table>

Reserved Instruction16

Reserved Instruction16		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x00000007 Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000014 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction17

Reserved Instruction17		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x00000007 Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000035 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction18

Reserved Instruction18		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x0000000B Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000007 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction19

Reserved Instruction19		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x0000000B Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000006 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction20

Reserved Instruction20

Length Bias:	2

DWord	Bit	Description
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x0000000B Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000003 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction21

Reserved Instruction21

Length Bias: 2

DWord	Bit	Description
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x0000000B Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000002 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction22

Reserved Instruction22

Length Bias: 2

DWord	Bit	Description
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x0000000B Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000001 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction23

Reserved Instruction23				
Length Bias:	2			
DWord	**Bit**	**Description**		
0	31:29	**Opcode 0**		
			Default Value:	0x00000003
			Format:	Opcode
	28:27	**Opcode 1**		
			Default Value:	0x00000002
			Format:	Opcode
	26:23	**Opcode 2**		
			Default Value:	0x0000000B
			Format:	Opcode
	22:16	**Opcode 3**		
			Default Value:	0x00000005
			Format:	Opcode
	15:12	**Reserved**		
			Access:	RO
			Format:	MBZ
	11:0	**DWord Count**		
			Format:	=n
0..n	31:0	**Unknown Bitfield**		

Reserved Instruction24

Reserved Instruction24		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x0000000B Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000000 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction25

Reserved Instruction25		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x0000000B Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000021 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction26

Reserved Instruction26		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x0000000B Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000020 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction27

Reserved Instruction27		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x0000000B Format: Opcode
	22:16	**Opcode 3** Default Value: 0x00000004 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction28

Reserved Instruction28		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x00000001 Format: Opcode
	22:21	**Opcode 3** Default Value: 0x00000000 Format: Opcode
	20:16	**Opcode 4** Default Value: 0x00000005 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction29

Reserved Instruction29		
Length Bias:	1	
DWord	Bit	Description
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x00000001 Format: Opcode
	22:21	**Opcode 3** Default Value: 0x00000000 Format: Opcode
	20:16	**Opcode 4** Default Value: 0x0000000C Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction30

Reserved Instruction30		
Length Bias: 2		
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x00000001 Format: Opcode
	22:16	**Opcode 3** Default Value: 0x0000000B Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction31

Reserved Instruction31		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x00000001 Format: Opcode
	22:21	**Opcode 3** Default Value: 0x00000000 Format: Opcode
	20:16	**Opcode 4** Default Value: 0x0000000A Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction32

Reserved Instruction32

Length Bias: 2

DWord	Bit	Description
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x00000001 Format: Opcode
	22:21	**Opcode 3** Default Value: 0x00000000 Format: Opcode
	20:16	**Opcode 4** Default Value: 0x00000009 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction33

Reserved Instruction33		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:23	**Opcode 2** Default Value: 0x00000001 Format: Opcode
	22:21	**Opcode 3** Default Value: 0x00000000 Format: Opcode
	20:16	**Opcode 4** Default Value: 0x00000008 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Reserved Instruction34

Reserved Instruction34		
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Opcode 0** Default Value: 0x00000003 Format: Opcode
	28:27	**Opcode 1** Default Value: 0x00000002 Format: Opcode
	26:24	**Opcode 2** Default Value: 0x00000004 Format: Opcode
	23:21	**Opcode 3** Default Value: 0x00000000 Format: Opcode
	20:16	**Opcode 4** Default Value: 0x00000003 Format: Opcode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Count** Format: =n
0..n	31:0	**Unknown Bitfield**

Return

ret - Return

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

Return execution to the code sequence that called a subroutine. The ret instruction can be predicated or non-predicated. If non-predicated, all channels jump to the return IP in the first channel of src0 and restore CallMask from the second channel of src0. If predicated, the enabled channels jump to the return IP from the first channel of src0 and the corresponding bits in the CallMask are cleared to zero; if all CallMask bits are zero after the ret instruction, then execution jumps to the return IP from the first channel of src0. When SPF is on, the predication control must be scalar.

Format:

```
[(pred)] ret (exec_size) null src0
```

Restriction

This instruction cannot take accumulator as source.

Syntax

```
[(pred)] ret (exec_size) null reg
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         PcIP[n] = src0.chan[0];
         CallMask[n] = 0;
     } else {
         PcIP[n] = IP + 1;
     }
 }
 for ( n = exec_size; n < 32; n++ ) {
     PcIP[n] = IP + 1;
 }
 if ( CallMask[n:0] == 0) ) {  // all channels are zero
     Jump(src0.chan[0]);
     CallMask = src0.chan[1];
 }
```

DWord	Bit	Description	
0..3	127:96	**Reserved**	
		Exists If:	([Src0.IsImm]==false)
		Format:	MBZ

ret - Return

	127:96	**JIP** Exists If: ([Src0.IsImm]==true) Format: S31 The byte-aligned jump distance if a jump is taken for the channel
	95:80	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	95:64	**Reserved** Exists If: ([Src0.IsImm]==true) Format: MBZ
	79:66	**Src0.Operand** Exists If: ([Src0.IsImm]==false) Format: **DirectOperand**
	65:64	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	63:50	**Dst.Operand** Format: **DirectOperand**
	49:47	**Reserved** Access: RO Format: MBZ
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value. **Value** | **Name** 0 | false 1 | true
	45:34	**Reserved** Access: RO Format: MBZ
	33	**BranchCtrl** This field is used by ***goto***, **if**, and ***else*** instructions to control branching. See the **goto** instruction description for more information about BranchCtrl.
	32	**AtomicCtrl** Format: **AtomicCtrl**
	31	**MaskCtrl** Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".

ret - Return

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr>
<tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr>
</table>
</td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b>
<table>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table>
</td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table>
</td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td><b>PredCtrl</b></td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the</td></tr>
</table>

ret - Return		
		destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: **ChanOff** This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Rotate Left

rol - Rotate Left

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	false
Source Modifier:	false

Perform component-wise logical rotate left operation of the bits in src0 by the rotate count indicated in src1, storing the result in dst. src0 and src1 are treated as unsigned numbers with only the bits within the specified datatype used during this operation. This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers are supported. Extra precision bits available in accumulator are ignored during this operation and only the bits within the specified datatype are used.
src0 and dst must be of same datatype precision.

Format:

```
 [(pred)] rol[.cmod] (exec_size) dst src0 src1
```

Syntax

```
[(pred)] rol[.cmod] (exec_size) reg reg reg
 [(pred)] rol[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         RotateMask = sizeof(src0) * 8 - 1;
         dst.chan[n] = (src0.chan[n] « (src1.chan[n] & RotateMask)) |
                       src0.chan[n] » (-src1.chan[n] & RotateMask));
     }
 }
```

Src Types	Dst Types
UW, UD	UW, UD

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)
	125:122	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ

rol - Rotate Left

	121:120	**Src1.Mod**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**SrcMod**
	119:116	**Src1.VertStride**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**VertStride**
	115:113	**Src1.Width**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**Width**
	112	**Src1.AddrMode**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**AddrMode**
	111:98	**Src1.Operand**	
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
		Format:	**IndirectOperand**
	111:98	**Src1.Operand**	
		Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
		Format:	**DirectOperand**
	97:96	**Src1.HorzStride**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**HorzStride**
	95:92	**CondCtrl**	
		Format:	**FlagModifier**
	91:88	**Src1.DataType**	
		Exists If:	([Src1.IsImm]==true)
		Format:	**ImmDataType**
	91:88	**Src1.DataType**	
		Exists If:	([Src1.IsImm]==false)
		Format:	**RegDataType**
	87:84	**Src0.VertStride**	
		Format:	**VertStride**
	83:81	**Src0.Width**	
		Format:	**Width**
	80	**Src0.AddrMode**	
		Format:	**AddrMode**

<table>
<tr><th colspan="3">rol - Rotate Left</th></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src1.IsImm</b>
This field indicate that Source 1 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">rol - Rotate Left</th></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr></table></td></tr>
</table>

rol - Rotate Left

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td><b>Header</b></td></tr></table></td></tr>
</table>

Rotate Right

ror - Rotate Right

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	false
Source Modifier:	false

Perform component-wise logical rotate right operation of the bits in src0 by the rotate count indicated in src1, storing the result in dst. src0 and src1 are treated as unsigned numbers with only the bits within the specified datatype used during this operation. This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers are supported. Extra precision bits available in accumulator are ignored during this operation and only the bits within the specified datatype are used.

src0 and dst must be of same datatype precision.

Format:

```
 [(pred)] ror[.cmod] (exec_size) dst src0 src1
```

Syntax

```
[(pred)] ror[.cmod] (exec_size) reg reg reg
 [(pred)] ror[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         RotateMask = sizeof(src0) * 8 - 1;
         dst.chan[n] = (src0.chan[n] » (src1.chan[n] & RotateMask)) |
                       (src0.chan[n] « (-src1.chan[n] & RotateMask));
     }
 }
```

Src Types	Dst Types
UW, UD	UW, UD

DWord	Bit	Description	
0..3	127:126	**Reserved**	
		Exists If:	([Src1.IsImm]==false)
		Format:	MBZ
	127:96	**Src1.ImmValue[31:0]**	
		Exists If:	([Src1.IsImm]==true)
	125:122	**Reserved**	
		Exists If:	([Src1.IsImm]==false)
		Format:	MBZ

ror - Rotate Right

	121:120	**Src1.Mod**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**SrcMod**
	119:116	**Src1.VertStride**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**VertStride**
	115:113	**Src1.Width**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**Width**
	112	**Src1.AddrMode**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**AddrMode**
	111:98	**Src1.Operand**		
			Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
			Format:	**IndirectOperand**
	111:98	**Src1.Operand**		
			Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
			Format:	**DirectOperand**
	97:96	**Src1.HorzStride**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**HorzStride**
	95:92	**CondCtrl**		
			Format:	**FlagModifier**
	91:88	**Src1.DataType**		
			Exists If:	([Src1.IsImm]==true)
			Format:	**ImmDataType**
	91:88	**Src1.DataType**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**RegDataType**
	87:84	**Src0.VertStride**		
			Format:	**VertStride**
	83:81	**Src0.Width**		
			Format:	**Width**
	80	**Src0.AddrMode**		
			Format:	**AddrMode**

<table>
<tr><th colspan="3">ror - Rotate Right</th></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src1.IsImm</b>
This field indicate that Source 1 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
</table>

ror - Rotate Right

<table>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr></table></td></tr>
</table>

ror - Rotate Right

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td><b>Header</b></td></tr></table></td></tr>
</table>

Round Down

rndd - Round Down

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

The rndd instruction takes component-wise floating point downward rounding (to the integral float number closer to negative infinity) of src0 and storing the rounded integral float results in dst. This is commonly referred to as the floor() function. Each result follows the rules in the following tables based on the floating-point mode.

Format:

```
        [(pred)] rndd[.cmod] (exec_size) dst src0
```

Syntax

```
[(pred)] rndd[.cmod] (exec_size) reg reg
 [(pred)] rndd[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = floor(src0.chan[n]);
     }
 }
```

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true)
	95:92	**CondCtrl** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: **FlagModifier**
	95:64	**Src0.ImmValue[63:32]** Exists If: ([Src0.IsImm]==true) AND (([Src0.DataType]==:q) OR ([Src0.DataType]==:uq) OR ([Src0.DataType]==:df))

rndd - Round Down

	87:84	**Src0.VertStride** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: VertStride
	83:81	**Src0.Width** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: Width
	80	**Src0.AddrMode** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: AddrMode
	79:66	**Src0.Operand** Exists If: (([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Direct) Format: DirectOperand
	79:66	**Src0.Operand** Exists If: (([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Indirect) Format: IndirectOperand
	65:64	**Src0.HorzStride** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: HorzStride
	63:50	**Dst.Operand** Exists If: ([Dst.AddrMode]==Indirect) Format: IndirectOperand
	63:50	**Dst.Operand** Exists If: ([Dst.AddrMode]==Direct) Format: DirectOperand
	49:48	**Dst.HorzStride** Format: HorzStride
	47	**Reserved** Access: RO Format: MBZ

rndd - Round Down

<table>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td><b>Saturate</b></td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td><b>AccWrCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations</td></tr>
</table>

<table>
<tr><th colspan="3">rndd - Round Down</th></tr>
<tr><td></td><td></td><td>supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
</table>

rndd - Round Down

	15:0	**Header** Format: **Header**

Round to Nearest or Even

rnde - Round to Nearest or Even

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

The rnde instruction takes component-wise floating point round-to-even operation of src0 with results in two pieces - a downward rounded integral float results stored in dst and the round-to-even increments stored in the rounding increment bits. The round-to-even increment must be added to the results in dst to create the final round-to-even values to emulate the round-to-even operation, commonly known as the round() function. The final results are the one of the two integral float values that is nearer to the input values. If neither possibility is nearer, the even alternative is chosen. Each result follows the rules in the following tables based on the floating-point mode.

Format:

```
[(pred)] rnde[.cmod] (exec_size) dst src0
```

Syntax

```
[(pred)] rnde[.cmod] (exec_size) reg reg
 [(pred)] rnde[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         if ( src0.chan[n] - floor(src0.chan[n]) > 0.5f ) {
             dst.chan[n] = floor(src0.chan[n]) + 1;
         } else if ( src0.chan[n] - floor(src0.chan[n]) < 0.5f ) {
             dst.chan[n] = floor(src0.chan[n]);
         } else {
             if ( floor(src0.chan[n]) is odd ) {
                 dst.chan[n] = floor(src0.chan[n]) + 1;
             } else {
                 dst.chan[n] = floor(src0.chan[n]);
             }
         }
     }
 }
```

Src Types	Dst Types
F	F

DWord	Bit	Description	
0..3	127:96	**Src0.ImmValue[31:0]**	
		Exists If:	([Src0.IsImm]==true)

rnde - Round to Nearest or Even

	Bits	Field	Exists If	Format
	95:92	**CondCtrl**	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	**FlagModifier**
	95:64	**Src0.ImmValue[63:32]**	([Src0.IsImm]==true) AND (([Src0.DataType]==:q) OR ([Src0.DataType]==:uq) OR ([Src0.DataType]==:df))	
	87:84	**Src0.VertStride**	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	**VertStride**
	83:81	**Src0.Width**	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	**Width**
	80	**Src0.AddrMode**	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	**AddrMode**
	79:66	**Src0.Operand**	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Direct)	**DirectOperand**
	79:66	**Src0.Operand**	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Indirect)	**IndirectOperand**
	65:64	**Src0.HorzStride**	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	**HorzStride**
	63:50	**Dst.Operand**	([Dst.AddrMode]==Indirect)	**IndirectOperand**
	63:50	**Dst.Operand**	([Dst.AddrMode]==Direct)	**DirectOperand**

<table>
<tr><th colspan="3">rnde - Round to Nearest or Even</th></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td>HorzStride</td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td>SrcMod</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td>ImmDataType</td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
</table>

<table>
<tr><th colspan="3">rnde - Round to Nearest or Even</th></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b>
<table><tr><td>Format:</td><td>MBZ</td></tr></table>
Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>
This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
</table>

rnde - Round to Nearest or Even		
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Round to Zero

rndz - Round to Zero

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

The rndz instruction takes component-wise floating point round-to-zero operation of src0 with results in two pieces - a downward rounded integral float results stored in dst and the round-to-zero increments stored in the rounding increment bits. The round-to-zero increment must be added to the results in dst to create the final round-to-zero values to emulate the round-to-zero operation, commonly known as the truncate() function. The final results are the one of the two closest integral float values to the input values that is nearer to zero.

Format:

```
[(pred)] rndz[.cmod] (exec_size) dst src0
```

Syntax

```
[(pred)] rndz[.cmod] (exec_size) reg reg
 [(pred)] rndz[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = floor(src0.chan[n]);
         if ( abs(src0.chan[n]) < abs(dst.chan[n]) ) {
             dst.chan[n] = floor(src0.chan[n]) + 1;
         } else {
             dst.chan[n] = floor(src0.chan[n]);
         }
     }
 }
```

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true)

rndz - Round to Zero

	Bits	Field	Exists If	Format
	95:92	**CondCtrl**	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	**FlagModifier**
	95:64	**Src0.ImmValue[63:32]**	([Src0.IsImm]==true) AND (([Src0.DataType]==:q) OR ([Src0.DataType]==:uq) OR ([Src0.DataType]==:df))	
	87:84	**Src0.VertStride**	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	**VertStride**
	83:81	**Src0.Width**	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	**Width**
	80	**Src0.AddrMode**	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	**AddrMode**
	79:66	**Src0.Operand**	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Direct)	**DirectOperand**
	79:66	**Src0.Operand**	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Indirect)	**IndirectOperand**
	65:64	**Src0.HorzStride**	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))	**HorzStride**
	63:50	**Dst.Operand**	([Dst.AddrMode]==Indirect)	**IndirectOperand**
	63:50	**Dst.Operand**	([Dst.AddrMode]==Direct)	**DirectOperand**

<table>
<tr><th colspan="3">rndz - Round to Zero</th></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td>HorzStride</td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td>SrcMod</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td>ImmDataType</td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
</table>

rndz - Round to Zero

29 **CmptCtrl**

Format:	MBZ

Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.

Value	Name	Description
0	NoCompaction **[Default]**	No compaction. 128-bit native instruction supporting all instruction options.
1	Compacted	Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.

28 **PredInv**

This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields

Value	Name	Description
0	Positive **[Default]**	Positive polarity of predication. Use the predication mask produced by PredCtrl.
1	Negative	Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.

27:24 **PredCtrl**

Format:	**PredCtrl**

This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.

23 **FlagRegNum[0]**

This field specifies bit[0] of the register number for a flag register operand.

22 **FlagSubRegNum**

This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.

21:19 **ChanOff**

Format:	**ChanOff**

This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.

rndz - Round to Zero

	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Round Up

rndu - Round Up

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

The rndu instruction takes component-wise floating point upward rounding (to the integral float number closer to positive infinity) of src0, commonly known as the ceiling() function. Each result follows the rules in the following tables based on the floating-point mode.

Format:

```
        [(pred)] rndu[.cmod] (exec_size) dst src0
```

Syntax

```
[(pred)] rndu[.cmod] (exec_size) reg reg
 [(pred)] rndu[.cmod] (exec_size) reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         if ( src0.chan[n] - floor(src0.chan[n]) > 0.0f ) {
             dst.chan[n] = floor(src0.chan[n]) + 1;
         } else {
             dst.chan[n] = src0.chan[n];
         }
     }
 }
```

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true)
	95:92	**CondCtrl** Exists If: ([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df)) Format: FlagModifier
	95:64	**Src0.ImmValue[63:32]** Exists If: ([Src0.IsImm]==true) AND (([Src0.DataType]==:q) OR ([Src0.DataType]==:uq) OR ([Src0.DataType]==:df))

rndu - Round Up

	87:84	**Src0.VertStride**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
			Format:	**VertStride**
	83:81	**Src0.Width**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
			Format:	**Width**
	80	**Src0.AddrMode**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
			Format:	**AddrMode**
	79:66	**Src0.Operand**	Exists If:	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Direct)
			Format:	**DirectOperand**
	79:66	**Src0.Operand**	Exists If:	(([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))) AND ([Src0.AddrMode]==Indirect)
			Format:	**IndirectOperand**
	65:64	**Src0.HorzStride**	Exists If:	([Src0.IsImm]==false) OR (([Src0.DataType]!=:q) AND ([Src0.DataType]!=:uq) AND ([Src0.DataType]!=:df))
			Format:	**HorzStride**
	63:50	**Dst.Operand**	Exists If:	([Dst.AddrMode]==Indirect)
			Format:	**IndirectOperand**
	63:50	**Dst.Operand**	Exists If:	([Dst.AddrMode]==Direct)
			Format:	**DirectOperand**
	49:48	**Dst.HorzStride**	Format:	**HorzStride**
	47	**Reserved**	Access:	RO
			Format:	MBZ
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value.		

<table>
<tr><th colspan="3">rndu - Round Up</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td>SrcMod</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td>ImmDataType</td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.</td></tr>
</table>

rndu - Round Up

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table>
</td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table>
</td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td><b>PredCtrl</b></td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table>
<tr><td>Format:</td><td><b>ChanOff</b></td></tr>
</table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table>
<tr><td>Format:</td><td><b>ExecSize</b></td></tr>
</table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b>
<table>
<tr><td>Format:</td><td><b>Header</b></td></tr>
</table>
</td></tr>
</table>

Sampler Cache Media Block Read MSD

MSD_SC_MB - Sampler Cache Media Block Read MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Reserved** Access: RO Format: MBZ
	17:14	**Message Type** Default Value: 05h Format: Opcode Media Block Read Sampler Cache message
	13:11	**Reserved** Access: RO Format: MBZ
	10:8	**Vertical Line Stride Override** Format: MDC_VLSO If enabled, specifies the Vertical Line Stride and Vertical Line Stride Offset override fields.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Sampler Cache Oword Aligned Block Read MSD

MSD_SC_OWAB - Sampler Cache Oword Aligned Block Read MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Reserved** Access: RO Format: MBZ
	17:14	**Message Type** Default Value: 04h Format: Opcode Oword Aligned Block Read Sampler Cache message
	13:11	**Reserved** Access: RO Format: MBZ
	10:8	**Data Elements** Format: MDC_DB_OW Specifies the number of contiguous Owords to be read
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Scratch Block Read MSD

MSD0R_SB - Scratch Block Read MSD

Source:	EuSubFunctionDataPort0
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Scratch Block Message** Default Value: 1h Format: Opcode Scratch Block Message
	17	**Operation Type** Default Value: 0h Format: Opcode Scratch Block Read message
	16	**Reserved** Access: RO Format: MBZ
	15	**Invalidate After Read** Format: MDC_IAR Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	14	**Reserved** Access: RO Format: MBZ

MSD0R_SB - Scratch Block Read MSD

	13:12	**Data Elements** Format: | MDC_DB_HW Specifies the number of registers to be read or written
	11:0	**Address Offset** Format: | GeneralStateOffset[16:5] HWORD (32 byte) based address offset to the BufferAddress in the Message Header.

Scratch Block Write MSD

MSD0W_SB - Scratch Block Write MSD

Source:	EuSubFunctionDataPort0
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header.
	18	**Scratch Block Message** Default Value: 1h Format: Opcode Scratch Block Message
	17	**Operation Type** Default Value: 1h Format: Opcode Scratch Block Write message
	16:14	**Reserved** Access: RO Format: MBZ
	13:12	**Data Elements** Format: MDC_DB_HW Specifies the number of registers to be read or written
	11:0	**Address Offset** Format: GeneralStateOffset[17:6] HWORD (32 byte) based address offset to the BufferAddress in the Message Header.

Select

sel - Select

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

The sel instruction selectively moves the components in src0 or src1 into the channels of dst based on the predication. On a channel by channel basis, if the channel condition is true, data in src0 is moved into dst. Otherwise, data in src1 is moved into dst.

As the predication is used to select the two sources, it is not included in the evaluation of WrEn. The predicate clause is mandatory if cmod is omitted/0000b. If both predication and the conditional modifier are omitted, the results are undefined.

If the conditional modifier is specified (not 0000b, a compare is performed and the resulting condition flag is used for the sel instruction. Conditional modifiers .ge and .lt follow the cmpn rules, and all other conditional modifiers follow the cmp rules. Predication is not allowed in this mode.

A sel instruction with cmod .lt is used to emulate a MIN instruction.

A sel instruction with cmod .ge is used to emulate a MAX instruction.

For a sel instruction with a .lt or .ge conditional modifier, if one source is NaN and the other not NaN, the non-NaN source is the result. If both sources are NaNs, the result is NaN. For all other conditional modifiers, if either source is NaN then src1 is selected.

A sel instruction without a conditional modifier always copies a denorm source value to a denorm destination value (in the manner of a raw move). This applies even if the source modifies are set on the *sel* instruction sources.

The sel instruction uses any conditional modifier internally and does not update the flag register if a conditional modifier is used.

A sel instruction with cmod will flush denorm to zero, depending on the denorm mode bit.

Format:

```
        (pred) sel[.cmod] (exec_size) dst src0 src1
```

Restriction

Pure bfloat operation is not supported.

Syntax

```
(pred) sel[.cmod] (exec_size) reg reg reg
 (pred) sel[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn, NoPMask);
 if (cmod == "0000") {  // no CMod Evaluate(PMask);
     for ( n = 0; n < exec_size; n++ ) {
         if ( WrEn.chan[n] ) {
             if ( PMask.channel[n] ) {
```

sel - Select

```
                dst.chan[n] = src0.chan[n];
            } else {
                dst.chan[n] = src1.chan[n];
            }
        }
    }
} else {  // with CMod Evaluate(CMod);
    for ( n = 0; n < exec_size; n++ ) {
        if ( WrEn.chan[n] ) {
            if ( CMod.chan[n] ) {
                dst.chan[n] = src0.chan[n];
            } else {
                dst.chan[n] = src1.chan[n];
            }
        }
    }
}
```

Src Types	Dst Types
*B,*W,*D	*B,*W,*D
F	F
HF	HF
BF, F	BF, F

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)
	125:122	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	121:120	**Src1.Mod** Exists If: ([Src1.IsImm]==false) Format: **SrcMod**
	119:116	**Src1.VertStride** Exists If: ([Src1.IsImm]==false) Format: **VertStride**
	115:113	**Src1.Width** Exists If: ([Src1.IsImm]==false) Format: **Width**

sel - Select

	112	**Src1.AddrMode**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**AddrMode**
	111:98	**Src1.Operand**		
			Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
			Format:	**IndirectOperand**
	111:98	**Src1.Operand**		
			Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
			Format:	**DirectOperand**
	97:96	**Src1.HorzStride**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**HorzStride**
	95:92	**CondCtrl**		
			Format:	**FlagModifier**
	91:88	**Src1.DataType**		
			Exists If:	([Src1.IsImm]==true)
			Format:	**ImmDataType**
	91:88	**Src1.DataType**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**RegDataType**
	87:84	**Src0.VertStride**		
			Format:	**VertStride**
	83:81	**Src0.Width**		
			Format:	**Width**
	80	**Src0.AddrMode**		
			Format:	**AddrMode**
	79:66	**Src0.Operand**		
			Exists If:	([Src0.AddrMode]==Direct)
			Format:	**DirectOperand**
	79:66	**Src0.Operand**		
			Exists If:	([Src0.AddrMode]==Indirect)
			Format:	**IndirectOperand**
	65:64	**Src0.HorzStride**		
			Format:	**HorzStride**

sel - Select		
	63:50	**Dst.Operand** Exists If: ([Dst.AddrMode]==Direct) Format: **DirectOperand**
	63:50	**Dst.Operand** Exists If: ([Dst.AddrMode]==Indirect) Format: **IndirectOperand**
	49:48	**Dst.HorzStride** Format: **HorzStride**
	47	**Src1.IsImm** This field indicate that Source 1 operand is carrying an immediate value. Value | Name 0 | false **[Default]** 1 | true
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value. Value | Name 0 | false **[Default]** 1 | true
	45:44	**Src0.Mod** Format: **SrcMod**
	43:40	**Src0.DataType** Exists If: ([Src0.IsImm]==false) Format: **RegDataType**
	43:40	**Src0.DataType** Exists If: ([Src0.IsImm]==true) Format: **ImmDataType**
	39:36	**Dst.DataType** Format: **RegDataType**
	35	**Dst.AddrMode** Format: **AddrMode**
	34	**Saturate** Format: **Saturate**
	33	**AccWrCtrl** Format: **AccWrCtrl**
	32	**AtomicCtrl** Format: **AtomicCtrl**

<table>
<tr><th colspan="3">sel - Select</th></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr>
<tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b>
<table>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table>
<tr><td>Format:</td><td>PredCtrl</td></tr>
</table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
</table>

sel - Select

	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: **ChanOff** This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: **Header**

Send Message

send - Send Message

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false
Subfunctions:	SFID[95:92]

Description
The send instruction performs data communication between a thread and external function units, including shared functions (Data Port Read, Data Port Write, URB, and Message Gateway) and some fixed functions (e.g. Thread Spawner, who also have an unique Shared Function ID). The send instruction adds an entry to the EU's message request queue. The request message is stored in a split pair of contiguous GRF registers. Typically the header and addresses in one block and the data in another, but this is not strictly necessary and null may be passed as either parameter. The response message, if present, will be returned to a block of contiguous GRF registers. The return GRF writes may be in any order depending on the external function units. <src0> and <src1> are the lead GRF registers for the first and second block of the request respectively. <dest> is the lead GRF register for response. The message descriptor field <desc> contains the Message Length (the number of consecutive GRF registers corresponding to src0) and the Response Length (the number of consecutive GRF registers). It also contains the header present bit, and the function control signals. The extend message descriptor field <ex_desc> contains the target function ID, the Extended Message Length (the number of consecutive GRF registers corresponding to src1) and the extended function control signals. WrEn is forwarded to the target function in the message sideband. The send instruction is the only way to terminate a thread. When the EOT (End of Thread) bit of <ex_desc> is set, it indicates the end of thread to the EU, the Thread Dispatcher and, in most cases, the parent fixed function.
The send instruction performs data communication between a thread and external function units, including shared functions (Sampler, Data Port Read, Data Port Write, URB, and Message Gateway) and some fixed functions (e.g. Thread Spawner, who also have an unique Shared Function ID). The send instruction adds an entry to the EU's message request queue. The request message is stored in a split pair of contiguous GRF registers. Typically the header and addresses in one block and the data in another, but this is not strictly necessary and null may be passed as either parameter. The response message, if present, will be returned to a block of contiguous GRF registers. The return GRF writes may be in any order depending on the external function units. <src0> and <src1> are the lead GRF registers for the first and second block of the request respectively. <dest> is the lead GRF register for response. The message descriptor field <desc> contains the Message Length (the number of consecutive GRF registers corresponding to src0) and the Response Length (the number of consecutive GRF registers). It also contains the header present bit, and the function control signals. The extend message descriptor field <ex_desc> contains the target function ID, the Extended Message Length (the number of consecutive GRF registers corresponding to src1) and the extended function control signals. WrEn is forwarded to the target function in the message sideband. The send instruction is the only way to terminate a thread. When the EOT (End of Thread) bit of <ex_desc> is set, it indicates the end of thread to the EU, the Thread Dispatcher and, in most cases, the parent fixed function.

send - Send Message

Message descriptor field <desc> can be a 32-bit immediate, imm32, or a 32-bit scalar register, <reg32a>. The 32-bit scalar register <reg32a> must be the leading dword of the address register. It should be in the form of a0.0.

<src0> is a GRF register. It serves as the leading GRF register of the request.
<src1> is a GRF register or a null register. It serves as the leading GRF register for the second block of the request when it is not a null register. It is required that the second block of GRFs does not overlap with the first block. If it is a null register the Extended Message Length must be 0.
<dest> serves for two purposes: to provide the leading GRF register location for the response message if present, and to provide parameters to form the channel-enable sideband signals.
<dest> signals whether there is a response to the message request. It can be either a null register or a GRF register.
If <dest> is null, there is no response to the request. Meanwhile, the Response Length field in <desc> must be 0. Certain types of message requests, such as memory write (store) through the Data Port, do not want response data from the function unit. If so, the posted destination operand can be null.
If <dest> is a GRF register, the register number is forwarded to the shared function. In this case, the target function unit must send one or more response message phases back to the requesting thread. The number of response message phases must match the Response Length field in <desc>, which of course cannot be zero. For some cases, it could be an empty return message. An empty return message is defined as a single phase message with all channel enables turned off.
The 16-bit channel enables of the message sideband are formed based on the WrEn. Interpretation of the channel-enable sideband signals is subject to the target external function. In general for a 'send' instruction with return messages, they are used as the destination dword write mask for the GRF registers starting at <dest>. For a message that has multiple return phases, the same set of channel enable signals applies to all the return phases.

Send a message stored in GRF locations starting at <src0> followed by <src1> to a shared function identified by <ex_desc> along with control from <desc> and <ex_desc> with a GRF writeback location at <dest>.

Format:

```
        [(pred)] send.<sfid> (exec_size) <dest> <src0> <src1> <ex_desc> <desc> {[EOT]}
```

Programming Notes

Send instruction execution (reading GRFs and sending out) is guaranteed to be in-order for a SharedFunction specified by SFID except for SLM. SLM SharedFunction is decoded as follows.
SLM = ((SFID==DC0) && (desc[18] == 0) && (desc[7:0]==0xFE)) ||
((SFID==DC1) && (desc[7:0]==0xFE)) ||
((SFID==DC2) && (desc[19] == 0)&&(desc[7]==0x1))

Restriction

Software must obey the following rules in signaling the end-of-thread using the send instruction: The posted destination operand must be null. No acknowledgement is allowed for the send instruction that signifies the end of thread. This is to avoid deadlock as the EU is expecting to free up the terminated thread's resource. A thread must terminate with a send instruction with message to a shared function on the output message bus; therefore, it cannot terminate with a send instruction with message to the NULL function. For example, a thread may terminate with a URB write message or a render cache write message. A root thread originated from the media (generic) pipeline must terminate with a send instruction with message to the Thread Spawner unit. A child thread should also terminate with a send to TS. Please refer to the Media Chapter for more detailed

send - Send Message

description.
Software must obey the following rules in signaling the end-of-thread using the send instruction: The posted destination operand must be null. No acknowledgement is allowed for the send instruction that signifies the end of thread. This is to avoid deadlock as the EU is expecting to free up the terminated thread's resource. A thread must terminate with a send instruction with message to a shared function on the output message bus; therefore, it cannot terminate with a send instruction with message to the sampler unit or the NULL function. For example, a thread may terminate with a URB write message or a render cache write message. A root thread originated from the media (generic) pipeline must terminate with a send instruction with message to the Thread Spawner unit. A child thread should also terminate with a send to TS. Please refer to the Media Chapter for more detailed description.
A send instruction cannot update accumulator registers.
EOT must NOT be set for the send instruction when using indirect register addressing mode.
An EOT send must use register space r112-r127 for sources. This is to enable loading of a new thread into the same slot while the message with EOT for current thread is pending dispatch.

Syntax

```
[(pred)] send.<sfid> (exec_size) reg  greg  reg  (imm32|reg32a) (imm32|reg32a) {[EOT]}
```

Pseudocode

```
Evaluate(WrEn);
 <MsgChEnable> = WrEn;
 <SourceReg0> = <src0>.RegNum;
 <SourceReg1> = <src1>.RegNum;
 MessageEnqueue(<MsgChEnable>, <ResponseReg>, <SourceReg0>, <SourceReg1>,<ex_desc>,
<dest>);
```

DWord	Bit	Description
0..3	127:124	**ExDesc[31:28]** Exists If: ([ExDesc.IsReg]==false) Format: **ExMsgDesc[31:28]**
	127:124	**Reserved** Exists If: ([ExDesc.IsReg]==true) Format: MBZ
	123:122	**Desc[31:30]** Exists If: ([Desc.IsReg]==false) Format: **MsgDesc[31:30]**
	123:113	**Reserved** Exists If: ([Desc.IsReg]==true) Format: MBZ

send - Send Message

	121:113	**Desc[19:11]**	Exists If:	([Desc.IsReg]==false)
			Format:	**MsgDesc[19:11]**
	112	**Reserved**	Access:	RO
			Format:	MBZ
	111:104	**Src1.RegNum**	Format:	**DirectOperand[13:6]**
	103:99	**Src1.Length**	Exists If:	([ExDesc.IsReg]==false)
			Format:	**ExMsgDesc[10:6]**
	103:99	**Src1.Length**	Exists If:	([ExDesc.IsReg]==true) AND ([ExBSO]==true)
	103:99	**Reserved**	Exists If:	([ExDesc.IsReg]==true) AND ([ExBSO]==false)
			Format:	MBZ
	98	**Src1.RegFile**	Format:	**DirectOperand[0]**
	97:96	**Reserved**	Exists If:	([ExDesc.IsReg]==true)
			Format:	MBZ
	97:96	**ExDesc[27:26]**	Exists If:	([ExDesc.IsReg]==false)
			Format:	**ExMsgDesc[27:26]**
	95:92	**SFID**	Format:	**SFID**
	91:81	**Reserved**	Exists If:	([Desc.IsReg]==true)
			Format:	MBZ
	91:81	**Desc[10:0]**	Exists If:	([Desc.IsReg]==false)
			Format:	**MsgDesc[10:0]**
	80	**Reserved**	Access:	RO
			Format:	MBZ

send - Send Message

79:72	**Src0.RegNum**		
		Format:	DirectOperand[13:6]
71	**MsgDesc[29]**		
		Exists If:	([Desc.IsReg]==false)
		Format:	MsgDesc[29:29]
71:67	**Reserved**		
		Exists If:	([Desc.IsReg]==true)
		Format:	MBZ
70:67	**Src0.Length**		
		Exists If:	([Desc.IsReg]==false)
		Format:	MsgDesc[28:25]
66	**Src0.RegFile**		
		Format:	DirectOperand[0]
65:64	**Reserved**		
		Exists If:	([ExDesc.IsReg]==true)
		Format:	MBZ
65:64	**ExDesc[25:24]**		
		Exists If:	([ExDesc.IsReg]==false)
		Format:	ExMsgDesc[25:24]
63:56	**Dst.RegNum**		
		Format:	DirectOperand[13:6]
55:51	**Dst.Length**		
		Exists If:	([Desc.IsReg]==false)
		Format:	MsgDesc[24:20]
55:51	**Reserved**		
		Exists If:	([Desc.IsReg]==true)
		Format:	MBZ
50	**Dst.RegFile**		
		Format:	DirectOperand[0]
49	**ExDesc.IsReg** This field indicates that the extended message descriptor is provided by the address register, selected by the AddrSubRegNum[3:1].		
		Value	**Name**
		0	false
		1	true

<table>
<tr><th colspan="3">send - Send Message</th></tr>
<tr><td></td><td>48</td><td><b>Desc.IsReg</b>
This field indicates that the message descriptor is provided by the address subregister a0.0.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>false</td></tr>
<tr><td>1</td><td>true</td></tr>
</table></td></tr>
<tr><td></td><td>47:43</td><td><b>Reserved</b>
<table>
<tr><td>Exists If:</td><td>([ExDesc.IsReg]==true)</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>47:35</td><td><b>ExDesc[23:11]</b>
<table>
<tr><td>Exists If:</td><td>([ExDesc.IsReg]==false)</td></tr>
<tr><td>Format:</td><td><b>ExMsgDesc[23:11]</b></td></tr>
</table></td></tr>
<tr><td></td><td>42:40</td><td><b>AddrSubRegNum[3:1]</b>
<table>
<tr><td>Exists If:</td><td>([ExDesc.IsReg]==true)</td></tr>
<tr><td>Format:</td><td><b>AddrSubRegNum[3:1]</b></td></tr>
</table></td></tr>
<tr><td></td><td>39</td><td><b>ExBSO</b>
<table>
<tr><td>Exists If:</td><td>([ExDesc.IsReg]==true)</td></tr>
</table>
This field indicate the Extended Bindless Surface Offset (ExBSO) mode.
When in ExBSO mode the BSO is extended to 26bits and occupies the whole of address register selected by AddrSubRegNum[3:1], the CPS and Src1.Length fields are taken as immediate values from instruction.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Legacy <b>[Default]</b></td></tr>
<tr><td>1</td><td>ExBSO</td></tr>
</table></td></tr>
<tr><td></td><td>38:36</td><td><b>Reserved</b>
<table>
<tr><td>Exists If:</td><td>([ExDesc.IsReg]==true)</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>35</td><td><b>Reserved</b>
<table>
<tr><td>Exists If:</td><td>([ExDesc.IsReg]==true) AND ([ExBSO]==false)</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>35</td><td><b>ExMsgDesc[11]</b>
<table>
<tr><td>Exists If:</td><td>([ExDesc.IsReg]==true) AND ([ExBSO]==true)</td></tr>
<tr><td>Format:</td><td><b>ExMsgDesc[11:11]</b></td></tr>
</table></td></tr>
<tr><td></td><td>34</td><td><b>EOT</b>
This field controls the termination of the thread. For a send instruction, if this field is set, EU will terminate the thread and also set the EOT bit in the message sideband. This field only applies to the send instruction. It is not present for other instructions.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Thread is not terminated</td></tr>
</table></td></tr>
</table>

send - Send Message

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>EOT</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>FusionCtrl</b>
This field provides explicit control for EU fusion lock-step execution. When this bit is set to 1b, the instruction is executed serially starting from the first EU to the last EU in the fused set.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Normal lockstep execution</td></tr><tr><td>1</td><td>Serialized execution</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr></table></td></tr>
</table>

send - Send Message

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td>PredCtrl</td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td>ChanOff</td></tr></table>This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td>ExecSize</td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td>Header</td></tr></table></td></tr>
</table>

Send Message Conditional

sendc - Send Message Conditional

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false
Subfunctions:	SFID[95:92]

The sendc instruction has the same behavior as the sends instruction except the following. sendc first checks the dependent threads inside the Thread Dependency Register. There are up to 8 dependent threads in the TDR register. The sendc instruction executes only when all the dependent threads in the TDR register are retired.

Wait for dependencies in the TDR Register to clear, then send a message stored in GRF locations starting at <src0> followed by <src1> to a shared function identified by <ex_desc> along with control from <desc> and <ex_desc> with a GRF writeback location at <dest>.

Format:

```
        [(pred)] sendc.<sfid> (exec_size) <dest> <src0> <src1> <ex_desc> <desc> {[EOT]}
```

Restriction

The sendc instruction has the same restrictions as the send instruction.

Syntax

```
[(pred)] sendc.<sfid> (exec_size) reg  greg  reg  (imm32|reg32a) (imm32|reg32a) {[EOT]}
```

Pseudocode

```
if (TDR[7] ... || TDR[2] || TDR[1] || TDR[0]) { wait; }
 Evaluate(WrEn);
 <MsgChEnable> = WrEn;
 MessageEnqueue(<MsgChEnable>, <ResponseReg>, <src0>.RegNum, <src1>.RegNum; ,<ex_desc>,
<dest>.RegNum);
```

DWord	Bit	Description
0..3	127:124	**ExDesc[31:28]** Exists If: ([ExDesc.IsReg]==false) Format: **ExMsgDesc[31:28]**
	127:124	**Reserved** Exists If: ([ExDesc.IsReg]==true) Format: MBZ

sendc - Send Message Conditional

	123:122	**Desc[31:30]**		
			Exists If:	([Desc.IsReg]==false)
			Format:	**MsgDesc[31:30]**
	123:113	**Reserved**		
			Exists If:	([Desc.IsReg]==true)
			Format:	MBZ
	121:113	**Desc[19:11]**		
			Exists If:	([Desc.IsReg]==false)
			Format:	**MsgDesc[19:11]**
	112	**Reserved**		
			Access:	RO
			Format:	MBZ
	111:104	**Src1.RegNum**		
			Format:	**DirectOperand[13:6]**
	103:99	**Src1.Length**		
			Exists If:	([ExDesc.IsReg]==false)
			Format:	**ExMsgDesc[10:6]**
	103:99	**Src1.Length**		
			Exists If:	([ExDesc.IsReg]==true) AND ([ExBSO]==true)
	103:99	**Reserved**		
			Exists If:	([ExDesc.IsReg]==true) AND ([ExBSO]==false)
			Format:	MBZ
	98	**Src1.RegFile**		
			Format:	**DirectOperand[0]**
	97:96	**Reserved**		
			Exists If:	([ExDesc.IsReg]==true)
			Format:	MBZ
	97:96	**ExDesc[27:26]**		
			Exists If:	([ExDesc.IsReg]==false)
			Format:	**ExMsgDesc[27:26]**
	95:92	**SFID**		
			Format:	**SFID**
	91:81	**Reserved**		
			Exists If:	([Desc.IsReg]==true)
			Format:	MBZ

sendc - Send Message Conditional

	91:81	**Desc[10:0]**		
			Exists If:	([Desc.IsReg]==false)
			Format:	**MsgDesc[10:0]**
	80	**Reserved**		
			Access:	RO
			Format:	MBZ
	79:72	**Src0.RegNum**		
			Format:	**DirectOperand[13:6]**
	71	**MsgDesc[29]**		
			Exists If:	([Desc.IsReg]==false)
			Format:	**MsgDesc[29:29]**
	71:67	**Reserved**		
			Exists If:	([Desc.IsReg]==true)
			Format:	MBZ
	70:67	**Src0.Length**		
			Exists If:	([Desc.IsReg]==false)
			Format:	**MsgDesc[28:25]**
	66	**Src0.RegFile**		
			Format:	**DirectOperand[0]**
	65:64	**Reserved**		
			Exists If:	([ExDesc.IsReg]==true)
			Format:	MBZ
	65:64	**ExDesc[25:24]**		
			Exists If:	([ExDesc.IsReg]==false)
			Format:	**ExMsgDesc[25:24]**
	63:56	**Dst.RegNum**		
			Format:	**DirectOperand[13:6]**
	55:51	**Dst.Length**		
			Exists If:	([Desc.IsReg]==false)
			Format:	**MsgDesc[24:20]**
	55:51	**Reserved**		
			Exists If:	([Desc.IsReg]==true)
			Format:	MBZ
	50	**Dst.RegFile**		
			Format:	**DirectOperand[0]**

sendc - Send Message Conditional

49 **ExDesc.IsReg**

This field indicates that the extended message descriptor is provided by the address register, selected by the AddrSubRegNum[3:1].

Value	Name
0	false
1	true

48 **Desc.IsReg**

This field indicates that the message descriptor is provided by the address subregister a0.0.

Value	Name
0	false
1	true

47:43 **Reserved**

Exists If:	([ExDesc.IsReg]==true)
Format:	MBZ

47:35 **ExDesc[23:11]**

Exists If:	([ExDesc.IsReg]==false)
Format:	**ExMsgDesc[23:11]**

42:40 **AddrSubRegNum[3:1]**

Exists If:	([ExDesc.IsReg]==true)
Format:	**AddrSubRegNum[3:1]**

39 **ExBSO**

Exists If:	([ExDesc.IsReg]==true)

This field indicates the Extended Bindless Surface Offset (ExBSO) mode.
When in ExBSO mode the BSO is extended to 26bits and occupies the whole of address register selected by AddrSubRegNum[3:1], the CPS and Src1.Length fields are taken as immediate values from instruction.

Value	Name
0	Legacy **[Default]**
1	ExBSO

38:36 **Reserved**

Exists If:	([ExDesc.IsReg]==true)
Format:	MBZ

35 **Reserved**

Exists If:	([ExDesc.IsReg]==true) AND ([ExBSO]==false)
Format:	MBZ

35 **ExMsgDesc[11]**

Exists If:	([ExDesc.IsReg]==true) AND ([ExBSO]==true)
Format:	**ExMsgDesc[11:11]**

sendc - Send Message Conditional

<table>
<tr><td></td><td>34</td><td><b>EOT</b>
This field controls the termination of the thread. For a send instruction, if this field is set, EU will terminate the thread and also set the EOT bit in the message sideband. This field only applies to the send instruction. It is not present for other instructions.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Thread is not terminated</td></tr>
<tr><td>1</td><td>EOT</td></tr>
</table></td></tr>
<tr><td></td><td>33</td><td><b>FusionCtrl</b>
This field provides explicit control for EU fusion lock-step execution. When this bit is set to 1b, the instruction is executed serially starting from the first EU to the last EU in the fused set.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Normal lockstep execution</td></tr>
<tr><td>1</td><td>Serialized execution</td></tr>
</table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b>
<table>
<tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr>
</table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr>
<tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b>
<table>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">sendc - Send Message Conditional</th></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b>
<table><tr><td>Format:</td><td><b>Header</b></td></tr></table></td></tr>
</table>

SFC_AVS_CHROMA_Coeff_Table

SFC_AVS_CHROMA_Coeff_Table

Source:	BSpec
Length Bias:	2

This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Format: OpCode **Value** | **Name** | **Description** Ah | Media Misc **[Default]** | Media MFX/VEBOX+SFC Mode
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 6h SFC_AVS CHROMA Coeff_Table Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 3Fh Excludes DWord (0,1) Format: =n Total Length - 2
1..64	2047:0	**AVS CHROMA Coefficient Table Body** Format: SFC_AVS_CHROMA_COEFF_TABLE_BODY[32]

SFC_AVS_LUMA_Coeff_Table

SFC_AVS_LUMA_Coeff_Table

Source:	BSpec
Length Bias:	2

This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Format: OpCode **Value** | **Name** | **Description** Ah | Media Misc **[Default]** | Media MFX/VEBOX+SFC Mode
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 5h SFC_AVS LUMA Coeff_Table Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 7Fh Excludes DWord (0,1) Format: =n Total Length - 2
1..128	4095:0	**AVS LUMA Coefficient Table Body** Format: SFC_AVS_LUMA_COEFF_TABLE_BODY[32]

SFC_AVS_STATE

<table>
<tr><th colspan="3">SFC_AVS_STATE</th></tr>
<tr><td colspan="3">Source: BSpec
Length Bias: 2</td></tr>
<tr><td colspan="3">This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table>
<tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline</b>
<table>
<tr><td>Default Value:</td><td>2h Media</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>26:23</td><td><b>Media Command Opcode</b>
<table>
<tr><td>Format:</td><td>OpCode</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>Ah</td><td>Media Misc <b>[Default]</b></td><td>[] Media MFX/VEBOX+SFC Modegen</td></tr>
</table></td></tr>
<tr><td></td><td>22:21</td><td><b>SubOpcodeA</b>
<table>
<tr><td>Default Value:</td><td>0h Common</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>20:16</td><td><b>SubOpcodeB</b>
<table>
<tr><td>Default Value:</td><td>2h SFC_AVS_STATE</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b>
<table>
<tr><td>Default Value:</td><td>2h Excludes DWord (0,1)</td></tr>
<tr><td>Format:</td><td>=n</td></tr>
</table>
Total Length - 2</td></tr>
<tr><td>1..3</td><td>95:0</td><td><b>AVS State Body</b>
<table>
<tr><td>Format:</td><td>SFC_AVS_STATE_BODY</td></tr>
</table></td></tr>
</table>

SFC_FRAME_START

<table>
<tr><th colspan="3">SFC_FRAME_START</th></tr>
<tr><td colspan="3">Source: BSpec
Length Bias: 2</td></tr>
<tr><td colspan="3">This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table>
<tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline</b>
<table>
<tr><td>Default Value:</td><td>2h Media</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>26:23</td><td><b>Media Command Opcode</b>
<table>
<tr><td>Format:</td><td>OpCode</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>Ah</td><td>Media Misc <b>[Default]</b></td><td>Media MFX/VEBOX+SFC Mode</td></tr>
</table></td></tr>
<tr><td></td><td>22:21</td><td><b>SubOpcodeA</b>
<table>
<tr><td>Default Value:</td><td>0h Common</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>20:16</td><td><b>SubOpcodeB</b>
<table>
<tr><td>Default Value:</td><td>4h SFC_FRAME_START</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b>
<table>
<tr><td>Default Value:</td><td>0h Excludes DWord (0,1)</td></tr>
<tr><td>Format:</td><td>=n</td></tr>
</table>
Total Length - 2</td></tr>
<tr><td>1</td><td>31:0</td><td><b>Frame Start Body</b>
<table>
<tr><td>Format:</td><td>SFC_FRAME_START_BODY</td></tr>
</table></td></tr>
</table>

SFC_IEF_STATE

SFC_IEF_STATE

Source: BSpec

Length Bias: 2

This command is sent from VDBOX/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Format: OpCode **Value** | **Name** | **Description** Ah | Media Misc **[Default]** | Media MFX/VEBOX+SFC Mode
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 3h SFC_IEF_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 16h Excludes DWord (0,1) Format: =n Total Length - 2
1..23	735:0	**SFC IEF State Body** Format: SFC_IEF_STATE_BODY

SFC_LOCK

<table>
<tr><th colspan="3">SFC_LOCK</th></tr>
<tr><td colspan="3">Source: BSpec
Length Bias: 2</td></tr>
<tr><td colspan="3"><table><tr><th>Description</th></tr><tr><td>This command is used for VD/VE box to communicate with SFC before the start of any SFC workload. VD/VE uses this command to make sure that it has the ownership of SFC pipe before running workload with SFC since SFC is shared between VD/VE on a frame level.
For VD(MFX)-SFC workload, only decoder mode is allowed. Encoder mode cannot use SFC.</td></tr><tr><td>For VD(HCP)-SFC workload, only decoder mode is allowed. Encoder mode cannot use SFC</td></tr></table></td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline</b><table><tr><td>Default Value:</td><td>2h Media</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:23</td><td><b>Media Command Opcode</b><table><tr><td>Format:</td><td>OpCode</td></tr></table><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>Ah</td><td>Media Misc <b>[Default]</b></td><td>Media MFX/VEBOX+SFC Mode
For VD(MFX)+SFC mode, only decoder mode is allowed. Encoder mode cannot use SFC</td></tr></table></td></tr>
<tr><td></td><td>22:21</td><td><b>SubOpcodeA</b><table><tr><td>Default Value:</td><td>0h Common</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>20:16</td><td><b>SubOpcodeB</b><table><tr><td>Default Value:</td><td>0h SFC Lock</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b><table><tr><td>Default Value:</td><td>0h Excludes DWord (0,1)</td></tr><tr><td>Format:</td><td>=n</td></tr></table>Total Length - 2</td></tr>
</table>

SFC_LOCK		
1	31:0	**SFC Lock Body** Format: SFC_LOCK_BODY

SFC_STATE

SFC_STATE

Source:	BSpec
Length Bias:	2

This command is sent from VDBOX/HCP/VEBOX to SFC pipeline at the start of each frame once the lock request is granted.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Default Value: Ah Media MFX/VEBOX+SFC Mode Format: OpCode
	22:21	**SubOpcodeA** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcodeB** Default Value: 1h SFC_State Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Default Value: 2Bh Excludes DWord (0,1) Format: =n Total Length - 2
1..60	1919:0	**SFC State Body** Format: SFC_STATE_BODY

Shift Left

shl - Shift Left

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

Description

Restriction: Perform component-wise logical left shift of the bits in src0 by the shift count indicated in src1, storing the results in dst, inserting zero bits in the number of LSBs indicated by the shift count. Hardware detects overflow properly and uses it to perform any saturation operation on the result, as long as the shifted result is within 33 bits. Otherwise, the result is undefined. Note: For word and DWord operands, the accumulators have 33 bits.

In QWord mode, the shift count is taken from the low six bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 63. Otherwise, the shift count is taken from the low five bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 31. The operation uses QWord mode if src0 or dst has the Q or UQ type but not if src1 is the only operand with the Q or UQ type.

Format:

```
[(pred)] shl[.cmod] (exec_size) dst src0 src1
```

Restriction

Accumulator cannot be destination, implicit or explicit.

Syntax

```
[(pred)] shl[.cmod] (exec_size) reg reg reg
 [(pred)] shl[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         shiftCnt = src0 or dst has Q or UQ type ? src1.chan[n] & 0x3F : src1.chan[n] &
0x1F
         dst.chan[n] = src0.chan[n] « shiftCnt;
     }
 }
```

Src Types	Dst Types
*B,*W,*D	*B,*W,*D

DWord	Bit	Description

shl - Shift Left

0..3	127:126	**Reserved**		
			Exists If:	([Src1.IsImm]==false)
			Format:	MBZ
	127:96	**Src1.ImmValue[31:0]**		
			Exists If:	([Src1.IsImm]==true)
	125:122	**Reserved**		
			Exists If:	([Src1.IsImm]==false)
			Format:	MBZ
	121:120	**Src1.Mod**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**SrcMod**
	119:116	**Src1.VertStride**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**VertStride**
	115:113	**Src1.Width**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**Width**
	112	**Src1.AddrMode**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**AddrMode**
	111:98	**Src1.Operand**		
			Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
			Format:	**IndirectOperand**
	111:98	**Src1.Operand**		
			Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
			Format:	**DirectOperand**
	97:96	**Src1.HorzStride**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**HorzStride**
	95:92	**CondCtrl**		
			Format:	**FlagModifier**
	91:88	**Src1.DataType**		
			Exists If:	([Src1.IsImm]==true)
			Format:	**ImmDataType**

shl - Shift Left

<table>
<tr><td></td><td>91:88</td><td><b>Src1.DataType</b><table><tr><td>Exists If:</td><td>([Src1.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>87:84</td><td><b>Src0.VertStride</b><table><tr><td>Format:</td><td><b>VertStride</b></td></tr></table></td></tr>
<tr><td></td><td>83:81</td><td><b>Src0.Width</b><table><tr><td>Format:</td><td><b>Width</b></td></tr></table></td></tr>
<tr><td></td><td>80</td><td><b>Src0.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src1.IsImm</b>
This field indicate that Source 1 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">shl - Shift Left</th></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td><b>Saturate</b></td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td><b>AccWrCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">shl - Shift Left</th></tr>
<tr><td></td><td></td><td><table><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td>PredCtrl</td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b><table><tr><td>Format:</td><td>ChanOff</td></tr></table>This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b><table><tr><td>Format:</td><td>ExecSize</td></tr></table>This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b><table><tr><td>Format:</td><td>Header</td></tr></table></td></tr>
</table>

Shift Right

shr - Shift Right

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	true

Perform component-wise logical right shift with zero insertion of the bits in src0 by the shift count indicated in src1, storing the results in dst. Insert zero bits in the number of MSBs indicated by the shift count. When src0 is accumulator and/or source modifier is used with src0 the sign bit is inserted in the MSBs which comefrom the additional precision. **Note**: For Word and DWord operands, the accumulators have 33 bits.
Note: For unsigned src0 types, shr and asr produce the same result.

In QWord mode, the shift count is taken from the low six bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 63. Otherwise the shift count is taken from the low five bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 31. The operation uses QWord mode if src0 or dst has the Q or UQ type but not if src1 is the only operand with the Q or UQ type.

Format:

```
[(pred)] shr[.cmod] (exec_size) dst src0 src1
```

Syntax

```
[(pred)] shr[.cmod] (exec_size) reg reg reg
 [(pred)] shr[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         shiftCnt = src0 or dst has Q or UQ type ? src1.chan[n] & 0x3F : src1.chan[n] & 0x1F
         dst.chan[n] = src0.chan[n] » shiftCnt;
     }
 }
```

Src Types	Dst Types
UB, UW, UD	UB, UW, UD

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)

shr - Shift Right

125:122	**Reserved**	Exists If: ([Src1.IsImm]==false)	Format: MBZ
121:120	**Src1.Mod**	Exists If: ([Src1.IsImm]==false)	Format: **SrcMod**
119:116	**Src1.VertStride**	Exists If: ([Src1.IsImm]==false)	Format: **VertStride**
115:113	**Src1.Width**	Exists If: ([Src1.IsImm]==false)	Format: **Width**
112	**Src1.AddrMode**	Exists If: ([Src1.IsImm]==false)	Format: **AddrMode**
111:98	**Src1.Operand**	Exists If: ([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)	Format: **IndirectOperand**
111:98	**Src1.Operand**	Exists If: ([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)	Format: **DirectOperand**
97:96	**Src1.HorzStride**	Exists If: ([Src1.IsImm]==false)	Format: **HorzStride**
95:92	**CondCtrl**		Format: **FlagModifier**
91:88	**Src1.DataType**	Exists If: ([Src1.IsImm]==true)	Format: **ImmDataType**
91:88	**Src1.DataType**	Exists If: ([Src1.IsImm]==false)	Format: **RegDataType**
87:84	**Src0.VertStride**		Format: **VertStride**
83:81	**Src0.Width**		Format: **Width**

shr - Shift Right

80	**Src0.AddrMode**		
		Format:	**AddrMode**
79:66	**Src0.Operand**		
		Exists If:	([Src0.AddrMode]==Direct)
		Format:	**DirectOperand**
79:66	**Src0.Operand**		
		Exists If:	([Src0.AddrMode]==Indirect)
		Format:	**IndirectOperand**
65:64	**Src0.HorzStride**		
		Format:	**HorzStride**
63:50	**Dst.Operand**		
		Exists If:	([Dst.AddrMode]==Direct)
		Format:	**DirectOperand**
63:50	**Dst.Operand**		
		Exists If:	([Dst.AddrMode]==Indirect)
		Format:	**IndirectOperand**
49:48	**Dst.HorzStride**		
		Format:	**HorzStride**
47	**Src1.IsImm**		
	This field indicate that Source 1 operand is carrying an immediate value.		
		Value	**Name**
		0	false **[Default]**
		1	true
46	**Src0.IsImm**		
	This field indicate that Source 0 operand is carrying an immediate value.		
		Value	**Name**
		0	false **[Default]**
		1	true
45:44	**Src0.Mod**		
		Format:	**SrcMod**
43:40	**Src0.DataType**		
		Exists If:	([Src0.IsImm]==false)
		Format:	**RegDataType**
43:40	**Src0.DataType**		
		Exists If:	([Src0.IsImm]==true)
		Format:	**ImmDataType**

<table>
<tr><th colspan="3">shr - Shift Right</th></tr>
<tr><td></td><td>39:36</td><td><b>Dst.DataType</b><table><tr><td>Format:</td><td>RegDataType</td></tr></table></td></tr>
<tr><td></td><td>35</td><td><b>Dst.AddrMode</b><table><tr><td>Format:</td><td>AddrMode</td></tr></table></td></tr>
<tr><td></td><td>34</td><td><b>Saturate</b><table><tr><td>Format:</td><td>Saturate</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>AccWrCtrl</b><table><tr><td>Format:</td><td>AccWrCtrl</td></tr></table></td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td>AtomicCtrl</td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr>
<tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr>
<tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields</td></tr>
</table>

shr - Shift Right

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table>
</td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b>
<table><tr><td>Format:</td><td><b>Header</b></td></tr></table>
</td></tr>
</table>

Signal Event

<table>
<tr><th colspan="3">MSD_SIGNAL_EVENT - Signal Event</th></tr>
<tr><td colspan="3">Source: EuSubFunctionGateway
Length Bias: 1</td></tr>
<tr><td colspan="3">Sends an event to all threads that are monitoring that Event ID.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>28:25</td><td><b>Message Length</b>
<table>
<tr><td>Format:</td><td>U4</td></tr>
</table>
Specifies the number of GRF registers sent as the message payload.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1</td><td>One <b>[Default]</b></td><td>See MDP_EVENT Event Data payload definition.</td></tr>
</table></td></tr>
<tr><td></td><td>24:20</td><td><b>Response Length</b>
<table>
<tr><td>Default Value:</td><td>0 None</td></tr>
<tr><td>Format:</td><td>U5</td></tr>
</table>
Specifies the number of GRF registers expected as the message response payload.</td></tr>
<tr><td></td><td>19:3</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>2:0</td><td><b>Signal Event Subfunction</b>
<table>
<tr><td>Default Value:</td><td>0x1</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
</table>

SIMD8 Render Target Read MSD

MSD_RTR_SIMD8 - SIMD8 Render Target Read MSD

Source:	EuSubFunctionRenderDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message **Programming Notes** This field must be programmed to 0
	29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Reserved** Access: RO Format: MBZ
	17:14	**Message Type** Default Value: 0Dh Format: Opcode Render Target Read message
	13	**Per-Sample PS Enable** Format: Enable If set, PS reads color phases on per sample basis for each slot.

MSD_RTR_SIMD8 - SIMD8 Render Target Read MSD

		Programming Notes This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Reserved** Access: RO Format: MBZ
	11	**Slot Group Select** Format: MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
	10:9	**Reserved** Access: RO Format: MBZ
	8	**Render Target Message Subtype** Default Value: 1h Format: Opcode SIMD8 single source message. Use slots [7:0] for the pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

SIMD8 Render Target Write MSD

MSD_RTW_SIMD8 - SIMD8 Render Target Write MSD

Source:	EuSubFunctionRenderDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message
	29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Per-Coarse Pixel PS outputs enable** Format: Enable This bit indicates the render target write is a coarse pixel write. **Programming Notes** This bit must be DISABLED if a pixel shader thread is dispatched at pixel- or sample- rate. This bit may be DISABLED if a multi-rate pixel shader thread is dispatched at coarse rate. In such case, it indicates per-pixel or per-sample write (as determined by Per-Sample PS outputs enable) from a multi-rate shader, where the set of pixels is indicated by the Pixel shading phase field in Extended Message descriptor.
	17:14	**Message Type** Default Value: 0Ch Format: Opcode Render Target Write message

MSD_RTW_SIMD8 - SIMD8 Render Target Write MSD

<table>
<tr><td></td><td>13</td><td><b>Per-Sample PS Enable</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot.
<table><tr><th>Programming Notes</th></tr>
<tr><td>This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE.
When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0.
When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples).
When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.</td></tr></table>
</td></tr>
<tr><td></td><td>12</td><td><b>Last Render Target Select</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.
</td></tr>
<tr><td></td><td>11</td><td><b>Slot Group Select</b>
<table><tr><td>Format:</td><td>MDC_RT_SGS</td></tr></table>
This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
</td></tr>
<tr><td></td><td>10:8</td><td><b>Render Target Message Subtype</b>
<table><tr><td>Default Value:</td><td>4h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table>
SIMD8 single source message. Use slots [7:0] for pixel enables, X/Y addresses, and oMask.
<table><tr><th>Programming Notes</th></tr>
<tr><td>The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].</td></tr></table>
</td></tr>
<tr><td></td><td>7:0</td><td><b>Binding Table Index</b>
<table><tr><td>Format:</td><td>MDC_BTS</td></tr></table>
Specifies the Binding Table Index for the message
</td></tr>
</table>

SIMD16 Render Target Read MSD

MSD_RTR_SIMD16 - SIMD16 Render Target Read MSD

Source:	EuSubFunctionRenderDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message **Programming Notes** This field must be programmed to 0
	29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Reserved** Access: RO Format: MBZ
	17:14	**Message Type** Default Value: 0Dh Format: Opcode Render Target Read message
	13	**Per-Sample PS Enable** Format: Enable If set, PS reads color phases on per sample basis for each slot.

MSD_RTR_SIMD16 - SIMD16 Render Target Read MSD

<table>
<tr><td></td><td></td><td><table><tr><th>Programming Notes</th></tr><tr><td>This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE.
When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0.
When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples).
When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.</td></tr></table></td></tr>
<tr><td></td><td>12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11</td><td><b>Slot Group Select</b><table><tr><td>Format:</td><td>MDC_RT_SGS</td></tr></table>This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.</td></tr>
<tr><td></td><td>10:9</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>8</td><td><b>Render Target Message Subtype</b><table><tr><td>Default Value:</td><td>0h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table>SIMD16 single source message. Use slots [15:0] for the pixel enables, X/Y addresses, and oMask.<table><tr><th>Programming Notes</th></tr><tr><td>The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>Binding Table Index</b><table><tr><td>Format:</td><td>MDC_BTS</td></tr></table>Specifies the Binding Table Index for the message</td></tr>
</table>

SIMD16 Render Target Write MSD

MSD_RTW_SIMD16 - SIMD16 Render Target Write MSD

Source:	EuSubFunctionRenderDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Message Precision Subtype** Default Value: 0h Format: Opcode Full precision data message
	29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the 2-register header.
	18	**Per-Coarse Pixel PS outputs enable** Format: Enable This bit indicates the render target write is a coarse pixel write. **Programming Notes** This bit must be DISABLED if a pixel shader thread is dispatched at pixel- or sample- rate. This bit may be DISABLED if a multi-rate pixel shader thread is dispatched at coarse rate. In such case, it indicates per-pixel or per-sample write (as determined by Per-Sample PS outputs enable) from a multi-rate shader, where the set of pixels is indicated by the Pixel shading phase field in Extended Message descriptor. When this bit is set and the message has oMask present, oMask represents the pixel enables within the Coarse Pixel in the row major order.

MSD_RTW_SIMD16 - SIMD16 Render Target Write MSD

	17:14	**Message Type** Default Value: 0Ch Format: Opcode Render Target Write message
	13	**Per-Sample PS Enable** Format: Enable If set, PS sends Render Target Write Message that outputs color, depth(optional) and stencil(optional) phases on per sample basis for each slot. **Programming Notes** This bit must be set when PS runs at sample-frequency i.e. pixel shader dispatch mode is PER_SAMPLE. When this bit is set and PS runs at pixel-frequency, i.e. pixel shader dispatch mode is PER_PIXEL, Render Target read and write messages interpret bits 9:6 in MCH_RT_C0 as sample index. In this mode, render target write message payload and render target read writeback payload contain color of a specific sample in all dispatched pixels. RT writes referring to out-of-bound samples have no effect. RT reads from out-of-bound samples return 0. When this bit is clear and PS runs at pixel-frequency, render target write messages contain color value for entire pixel (all samples). When this bit is clear and PS runs at pixel-frequency, render target reads are disallowed per API spec (RT read without specifying sample index forces sample-frequency dispatch). HW behavior is undefined in such case.
	12	**Last Render Target Select** Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.
	11	**Slot Group Select** Format: MDC_RT_SGS This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
	10:8	**Render Target Message Subtype** Default Value: 0h Format: Opcode SIMD16 Single source message. Use slots [15:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

STATE_BASE_ADDRESS

STATE_BASE_ADDRESS

Source:	BSpec
Length Bias:	2

The STATE_BASE_ADDRESS command sets the base pointers for subsequent state, instruction, and media indirect object accesses by the GPE.
For more information see the Base Address Utilization table in the Memory Access Indirection narrative topic.

Programming Notes

The following commands must be reissued following any change to the base addresses:

- 3DSTATE_CC_POINTERS
- 3DSTATE_BINDING_TABLE_POINTERS
- 3DSTATE_SAMPLER_STATE_POINTERS
- 3DSTATE_VIEWPORT_STATE_POINTERS

Execution of this command causes a full pipeline flush, thus its use should be minimized for higher performance.

If 3DSTATE_PS_EXTRA::Pixel Shader Is Per Coarse Pixel == 1, the 3DSTATE_CPS_POINTERS command must be reissued following any change to the dynamic state base address.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h GFXPIPE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 01h STATE_BASE_ADDRESS Format: OpCode
	15:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Format: =n

<table>
<tr><th colspan="3">STATE_BASE_ADDRESS</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>14h</td><td>DWORD_COUNT_n <b>[Default]</b></td><td>Excludes DWord (0,1)</td></tr></table></td></tr>
<tr><td>1..2</td><td>63:12</td><td><b>General State Base Address</b><table><tr><td>Format:</td><td>GraphicsAddress[63:12]</td></tr></table><table><tr><th>Description</th></tr><tr><td>Specifies the 4K-byte aligned base address for general state accesses. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>Bounds checking is performed on general state accesses by Data Port Shared Functions for stateless A32 messages.
Bounds checking is enabled when General State Base Address [46:12] + General State Buffer Size [31:12] is <= 2^47. This ensures that the General State Buffer does not straddle the canonical address boundary where GraphicsAddress [47] changes.</td></tr></table><table><tr><th>Restriction</th></tr><tr><td>General State Base Address [47:12] + General State Buffer Size [31:12] must be < 2^48. It is illegal programming for this to be >= 2^48.</td></tr><tr><td>When using stateless (A32) Data Port messages, General State Base Address [47:12] + Buffer Base Address [31:0] must be < 2^48. It is illegal for this to be >= 2^48.</td></tr></table></td></tr>
<tr><td></td><td>11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10:4</td><td><b>General State Memory Object Control State</b><table><tr><td>Format:</td><td>MEMORY_OBJECT_CONTROL_STATE</td></tr></table>Specifies the memory object control state for indirect state using the <b>General State Base Address</b>, with the exception of the stateless data port accesses.</td></tr>
<tr><td></td><td>3:1</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>General State Base Address Modify Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>The other fields in this DWord and the following DWord are updated only when this bit is set.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Disable</td><td>Ignore the updated address.</td></tr><tr><td>1h</td><td>Enable</td><td>Modify the address.</td></tr></table></td></tr>
</table>

STATE_BASE_ADDRESS

3	31:26	**Reserved** Access: | RO Format: | MBZ
	25:23	**L1 Cache Control** Format: | L1_CACHE_CONTROL Specifies the Untyped L1 default cacheability attributes for stateless accesses.
	22:16	**Stateless Data Port Access Memory Object Control State** Format: | MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for stateless data port accesses.
	15:14	**Reserved** Access: | RO Format: | MBZ
	13	**Enable memory compression for All Stateless Accesses** Enable compression for stateless memory accesses. Value | Name 0 | Disabled **[Default]** 1 | Enabled
	12:1	**Reserved** Access: | RO Format: | MBZ
	0	**Coherency Setting Modify Enable** All the fields in this DW is only updated when this bit is set. Value | Name 1 | Enable write to this DW 0 | Disable write to this DW **[Default]**
4..5	63:12	**Surface State Base Address** Format: | GraphicsAddress[63:12] Specifies the 4K-byte aligned base address for binding table and surface state accesses. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	11	**Reserved** Access: | RO Format: | MBZ
	10:4	**Surface State Memory Object Control State** Format: | MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for indirect state using the **Surface State Base Address**.

<table>
<tr><th colspan="3">STATE_BASE_ADDRESS</th></tr>
<tr><td></td><td>3:1</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>0</td><td><b>Surface State Base Address Modify Enable</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
The other fields in this DWord and the following DWord are updated only when this bit is set.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable</td><td>Ignore the updated address.</td></tr>
<tr><td>1h</td><td>Enable</td><td>Modify the address.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Setting this bit to 1 in a batch buffer causes the resource streamer to stop; for performance reasons the SW should only place commands with this bit set in the ring buffer.</td></tr>
<tr><td>Before programming the Surface State Base Address, the RS must be disabled. Within a batch buffer where the RS is enabled, RS may be disabled thru a MI_RS_CONTROL command with Resource Streamer Control cleared prior to the STATE_BASE_ADDRESS with Surface State Base Address Modify Enable set and then re-enabled with another MI_RS_CONTROL with Resource Streamer Control set.</td></tr>
</table></td></tr>
<tr><td>6..7</td><td>63:12</td><td><b>Dynamic State Base Address</b>
<table>
<tr><td>Format:</td><td>GraphicsAddress[63:12]</td></tr>
</table>
<table>
<tr><th>Description</th></tr>
<tr><td>Specifies the 4K-byte aligned base address for sampler and viewport state accesses. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].</td></tr>
</table></td></tr>
<tr><td></td><td>11</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>10:4</td><td><b>Dynamic State Memory Object Control State</b>
<table>
<tr><td>Format:</td><td>MEMORY_OBJECT_CONTROL_STATE</td></tr>
</table>
Specifies the memory object control state for indirect state using the <b>Dynamic State Base Address</b>. Push constants defined in 3DSTATE_CONSTANT_(VS | GS | PS) commands do not use this control state, although they can use the corresponding base address. The memory object control state for push constants is defined within the command.</td></tr>
<tr><td></td><td>3:1</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">STATE_BASE_ADDRESS</th></tr>
<tr><td></td><td>0</td><td><b>Dynamic State Base Address Modify Enable</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
The other fields in this DWord and the following DWord are updated only when this bit is set.
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable</td><td>Ignore the updated address.</td></tr>
<tr><td>1h</td><td>Enable</td><td>Modify the address.</td></tr></table></td></tr>
<tr><td>8..9</td><td>63:12</td><td><b>Indirect Object Base Address</b>
<table><tr><td>Format:</td><td>GraphicsAddress[63:12]</td></tr></table>
Specifies the 4K-byte aligned base address for indirect object load in MEDIA_OBJECT command.</td></tr>
<tr><td></td><td>11</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10:4</td><td><b>Indirect Object Memory Object Control State</b>
<table><tr><td>Format:</td><td>MEMORY_OBJECT_CONTROL_STATE</td></tr></table>
Specifies the memory object control state for indirect objects using the <b>Indirect Object Base Address</b>.</td></tr>
<tr><td></td><td>3:1</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>Indirect Object Base Address Modify Enable</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
The other fields in this DWord and the following DWord are updated only when this bit is set.
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Disable</td><td>Ignore the updated address.</td></tr>
<tr><td>1h</td><td>Enable</td><td>Modify the address.</td></tr></table></td></tr>
<tr><td>10..11</td><td>63:12</td><td><b>Instruction Base Address</b>
<table><tr><td>Format:</td><td>GraphicsAddress[63:12]</td></tr></table>
Specifies the 4K-byte aligned base address for all EU instruction accesses.
GraphicsAddress[63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].</td></tr>
<tr><td></td><td>11</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>10:4</td><td><b>Instruction Memory Object Control State</b>
<table><tr><td>Format:</td><td>MEMORY_OBJECT_CONTROL_STATE</td></tr></table>
Specifies the memory object control state for EU instructions using the <b>Instruction Base Address</b>.</td></tr>
</table>

<table>
<tr><th colspan="3">STATE_BASE_ADDRESS</th></tr>
<tr><td></td><td>3:1</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>Instruction Base Address Modify Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>The other fields in this DWord and the following DWord are updated only when this bit is set.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Disable</td><td>Ignore the updated address.</td></tr><tr><td>1h</td><td>Enable</td><td>Modify the address.</td></tr></table></td></tr>
<tr><td>12</td><td>31:12</td><td><b>General State Buffer Size</b><table><tr><td>Format:</td><td>U20</td></tr></table><table><tr><td>This field specifies the size of the buffer in 4K pages. Any access that straddles or goes past the end of the buffer returns 0.
Note that BufferSize=0 indicates that there is no valid data in the buffer.</td></tr></table></td></tr>
<tr><td></td><td>11:1</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>General State Buffer Size Modify Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table>The fields in this DWord are updated only when this bit is set.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Disable</td><td>Ignore the updated bound.</td></tr><tr><td>1h</td><td>Enable</td><td>Modify the updated bound.</td></tr></table></td></tr>
<tr><td>13</td><td>31:12</td><td><b>Dynamic State Buffer Size</b><table><tr><td>Format:</td><td>U20</td></tr></table><table><tr><td>This field specifies the size of the buffer in 4K pages. Any access that straddles or goes past the end of the buffer returns 0.
Note that BufferSize=0 indicates that there is no valid data in the buffer.</td></tr></table></td></tr>
<tr><td></td><td>11:1</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>Dynamic State Buffer Size Modify Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table><table><tr><td>The fields in this DWord are updated only when this bit is set.</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">STATE_BASE_ADDRESS</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Disable</td><td>Ignore the updated bound.</td></tr><tr><td>1h</td><td>Enable</td><td>Modify the updated bound.</td></tr></table></td></tr>
<tr><td>14</td><td>31:12</td><td><b>Indirect Object Buffer Size</b><table><tr><td>Format:</td><td>U20</td></tr></table><table><tr><td>This field specifies the size of the buffer in 4K pages. Any access that straddles or goes past the end of the buffer returns 0.
Note that BufferSize=0 indicates that there is no valid data in the buffer.</td></tr></table></td></tr>
<tr><td></td><td>11:1</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>Indirect Object Buffer Size Modify Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table><table><tr><td>The fields in this DWord are updated only when this bit is set.</td></tr></table><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Disable</td><td>Ignore the updated bound.</td></tr><tr><td>1h</td><td>Enable</td><td>Modify the updated bound.</td></tr></table></td></tr>
<tr><td>15</td><td>31:12</td><td><b>Instruction Buffer Size</b><table><tr><td>Format:</td><td>U20</td></tr></table><table><tr><td>This field specifies the size of the buffer in 4K pages. Any access that straddles or goes past the end of the buffer returns 0.
Note that BufferSize=0 indicates that there is no valid data in the buffer.</td></tr></table></td></tr>
<tr><td></td><td>11:1</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>Instruction Buffer size Modify Enable</b><table><tr><td>Format:</td><td>Enable</td></tr></table><table><tr><td>The fields in this DWord are updated only when this bit is set.</td></tr></table><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Disable</td><td>Ignore the updated bound.</td></tr></table></td></tr>
</table>

STATE_BASE_ADDRESS

16..17	63:12	**Bindless Surface State Base Address** Format: GraphicsAddress[63:12] Specifies the 4K-byte aligned base address for bindless surface state accesses. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	11	**Reserved** Access: RO Format: MBZ
	10:4	**Bindless Surface State Memory Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for indirect state using the **Bindless Surface State Base Address**.
	3:1	**Reserved** Access: RO Format: MBZ
	0	**Bindless Surface State Base Address Modify Enable** Format: Enable **Description** The other fields in this DWord and the following two DWords are updated only when this bit is set. **Value** | **Name** | **Description** 0h | Disable | Ignore the updated address 1h | Enable | Modify the address
18	31:0	**Bindless Surface State Size** Format: GraphicsAddress[37:6] This field indicates the size-1 of the Bindless Surface State buffer in 64-Byte increments. Any SSO beyond this maximum size points to offset 0. Example: If the buffer contains 512 surface states, then this field must be programmed to 0x1FF (511 decimal).
19..20	63:12	**Bindless Sampler State Base Address** Format: GraphicsAddress[63:12] Specifies the 4K-byte aligned base address for bindless sampler state accesses. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	11	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">STATE_BASE_ADDRESS</th></tr>
<tr><td></td><td>10:4</td><td><b>Bindless Sampler State Memory Object Control State</b>
<table><tr><td>Format:</td><td>MEMORY_OBJECT_CONTROL_STATE</td></tr></table>
Specifies the memory object control state for indirect state using the <b>Bindless Sampler State Base Address</b>.</td></tr>
<tr><td></td><td>3:1</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td><b>Bindless Sampler State Base Address Modify Enable</b>
<table><tr><td>Format:</td><td>Enable</td></tr></table>
The other fields in this DWord and the following two DWords are updated only when this bit is set.
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Disable</td><td>Ignore the updated address</td></tr><tr><td>1h</td><td>Enable</td><td>Modify the address</td></tr></table></td></tr>
<tr><td>21</td><td>31:12</td><td><b>Bindless Sampler State Buffer Size</b>
<table><tr><td>Format:</td><td>U20</td></tr></table>
This field specifies the size of the buffer in 4K pages. Any access that goes beyond the end of the buffer (as defined by the Sampler State Buffer Size) will use an offset of 0.</td></tr>
<tr><td></td><td>11:0</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

STATE_COMPUTE_MODE

<table>
<tr><th colspan="3">STATE_COMPUTE_MODE</th></tr>
<tr><td colspan="3">Source: RenderCS, ComputeCS
Length Bias: 2</td></tr>
<tr><td colspan="3">This is a non-pipeline state command and is a general compute programming state that can be shared from the top to bottom of the pipeline.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table>
<tr><td>Default Value:</td><td>3h GFXPIPE</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:27</td><td><b>Command SubType</b>
<table>
<tr><td>Default Value:</td><td>0h GFXPIPE_COMMON</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>26:24</td><td><b>3D Command Opcode</b>
<table>
<tr><td>Default Value:</td><td>1h GFXPIPE_NONPIPELINED</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>23:16</td><td><b>3D Command Sub Opcode</b>
<table>
<tr><td>Default Value:</td><td>05h STATE_COMPUTE_MODE</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>15:8</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Default Value:</td><td>0h Excludes DWord (0,1)</td></tr>
<tr><td>Format:</td><td>=n</td></tr>
</table></td></tr>
<tr><td>1</td><td>31:16</td><td><b>Mask</b>
<table>
<tr><td>Format:</td><td>Enable[16]</td></tr>
</table>
This field is the mask bits for the state bits below. This is a bit wise mask where the bit number-16 is the value of the corresponding bit being masked in the same data word. For example, if you want to update state for bits 3:2 then bits 19:18 must be set.</td></tr>
<tr><td></td><td>15</td><td><b>Large GRF Mode</b>
<table>
<tr><td>This bit controls the Large GRF Mode Vs Regular GRF Mode in Execution Units.</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td><b>[Default]</b></td><td>Regular GRF mode of operation.</td></tr>
<tr><td>1</td><td></td><td>Large GRF mode of operation.</td></tr>
</table></td></tr>
</table>

STATE_COMPUTE_MODE

Programming Notes
Large GRF Mode bit functionality will take place in hardware only when the context is programmed to execute in Run Alone mode.

14 **Reserved**

Format:	MBZ

13 **Disable L1 Invalidate for non-L1-cacheable Writes**

Format:	Disable

When this bit is set, HDC global memory write requests that are marked "non-L1-cacheable" (either due to MOCS setting or L1-cache-disable mode bits set in this register) will not send "Invalidation" request to the SamplerL1 cache. The implication of this bit being set is that HDC will not maintain RAW and WAR ordering between L1-cacheable and non-L1-cacheable requests to the same address. URB writes never sends Invalidate commands to Sampler L1 cache (regardless of this bit).

Value	Name	Description
0b	Enable **[Default]**	HDC Non-L1-cacheable writes to Global memory will send Invalidate command to Sampler L1 cache.
1b	Disable	HDCNon-L1-cacheable writes to Global memory will NOT send Invalidate command to Sampler L1 cache.

12 **Reserved**

Access:	RO
Format:	MBZ

11 **Disable Atomic on Clear Data**

Setting this bit will disable HDC H/W support of Atomic operations on "fast-clear" compressed data. It will be the driver's responsibility to resolve clear data before any Atomic operation by HDC.

Value	Name	Description
1	Disable	Disable HDC support for Atomic operations on Clear data.
0	Enable **[Default]**	Enable HDC support of Atomic operations on Clear data.

10 **Reserved**

Access:	RO
Format:	MBZ

9:7 **Pixel Async Compute Thread Limit**

Specifies the maximum number of active Compute CS threads to run in a DSS when the 3D Pipe is active and a Z-pass is not running. When the 3D Pipe is not active or when a Z-pass is running, the maximum number of active Compute CS threads is specified by **Maximum Number of Threads** in CFE_STATE command.

Value	Name	Description	Programming Notes
0	Disabled **[Default]**	No limit applied. **Maximum Number of Threads** is the only limit on Compute CS threads.	

1	Max 2	Maximum of 2 Fused-EU threads per DSS, when 3D Pipe is active. This sets the Async Compute thread limit to about 1 thread per EU row.	
2	Max 8	Maximum of 8 Fused-EU threads per DSS, when 3D Pipe is active. This sets the Async Compute thread limit to about 1 thread per EU .	This value is the recommended SW setting, to balance forward progress on Async Compute and 3D Pipe dispatches.
3	Max 16	Maximum of 16 Fused-EU threads per DSS, when 3D Pipe is active. This sets the Async Compute thread limit to about 2 threads per EU .	
4	Max 24	Maximum of 24 Fused-EU threads per DSS, when 3D Pipe is active.	
5	Max 32	Maximum of 32 Fused-EU threads per DSS, when 3D Pipe is active.	
6	Max 40	Maximum of 40 Fused-EU threads per DSS, when 3D Pipe is active.	
7	Max 48	Maximum of 48 Fused-EU threads per DSS, when 3D Pipe is active. This sets the Async Compute thread limit to about half the threads per EU .	

6

Disable SLM Read Merge Optimization

Disable merging of Block Read Messages in SLM

Value	Name
0	Enabled **[Default]**
1	Disabled

5

Fast Clear Disabled on Compressed Surface

SW can set this bit if "fast-clear" state is not used for compressed surfaces. This allows optimization of memory partial writes for data port messages.

Value	Name
0	Enabled **[Default]**
1	Disabled

4:3

Force Non-Coherent

Format:	U2

Force all Data Cache Data Port access to be Non-Coherent (virtual addresses) and non-faultable regardless of the surface state or binding table index.

Value	Name	Description
0h	Force Disabled **[Default]**	GPU Coherence with CPU and Multi-GPU is computed normally based on surface state settings.

STATE_COMPUTE_MODE

1h	Force CPU Non-Coherent	GPU accesses are forced as non-coherent with CPU. GPU accesses to Multi-GPU are computed normally based on surface state settings.
2h	Force GPU Non-Coherent	GPU accesses are forced as non-coherent with other GPU, as well as with the CPU.

Programming Notes
Only change this mode after a pipeflush and cache flush (all threads and their all accesses completed).
CPU-GPU or GPU-GPU coherency is not supported. Hence the driver must set this field to 0x2 (Force GPU non-coherent), if the data-port message has coherency enabled via BTI or surface state.

2:0

Z Pass Async Compute Thread Limit

Specifies the maximum number of active Compute CS threads to run in a DSS when the 3D Pipe is active and a Z-pass is running.

Value	Name	Description	Programming Notes
0	Max 60 **[Default]**	Maximum of upto 1 thread per fused EU reserved for 3D.	This value is the recommended SW setting, to balance forward progress on Async Compute and 3D Pipe dispatches.
1	Max 64	No limit applied. **Maximum Number of Threads** is the only limit on Compute CS threads.	
2	Max 56	Maximum of 1 thread per fused EU reserved for 3D .	
3	Max 48	Maximum of 2 thread per fused EU reserved for 3D .	

STATE_SIP

<table>
<tr><th colspan="3">STATE_SIP</th></tr>
<tr><td colspan="3">Source: BSpec
Length Bias: 2</td></tr>
<tr><td colspan="3">The STATE_SIP command specifies the starting instruction location of the System Routine that is shared by all threads in execution.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b>
<table><tr><td>Default Value:</td><td>3h GFXPIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Command SubType</b>
<table><tr><td>Default Value:</td><td>0h GFXPIPE_COMMON</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><b>3D Command Opcode</b>
<table><tr><td>Default Value:</td><td>1h GFXPIPE_NONPIPELINED</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>3D Command Sub Opcode</b>
<table><tr><td>Default Value:</td><td>02h STATE_SIP</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:8</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table><tr><td>Format:</td><td>=n</td></tr></table>
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>1h</td><td>DWORD_COUNT_n <b>[Default]</b></td><td>Excludes DWord (0,1)</td></tr></table></td></tr>
<tr><td>1..2</td><td>63:4</td><td><b>System Instruction Pointer</b>
<table><tr><td>Format:</td><td>InstructionBaseOffset[63:4]</td></tr></table>
Specifies the instruction address of the system routine associated with the current context as a 128-bit granular offset from the Instruction Base Address. SIP is shared by all threads in execution. The address specifies the double quadword aligned instruction location.
GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
<table><tr><th>Programming Notes</th></tr><tr><td>This portion of the command is not context save/restored. The context image may restore this command as a 2 dword command rather than a 3 dword command.</td></tr></table></td></tr>
</table>

STATE_SIP		
	3:0	**Reserved** Access: RO Format: MBZ

Store

DP_STORE - Store

Source:	SFID_1, SFID_6, SFID_E, SFID_F
Length Bias:	1

Store untyped data to memory. For each enabled SIMT lane, a vector is written into memory from registers.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size. If not transposed, Vector Size specifies how many sequential copies of the data payload are in the message. If transposed, the Exec_Mask specifies how many sequential copies of the data payload are in the message.

Restriction	Source
Store is not supported by data port URB.	SFID_6

Syntax

```
[(pred)] STORE.sfid[.cache] (exec_mask) <addr_type[+offset]>src0_reg:addr_size
src1_reg:data_size[.vect_size][transpose]
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base;
for (n = 0; n < 32; n++) {
    if (Msg.ChEn[n]) {
        for (v = 0; v < vect_size; v++) {
            if (transpose) {
                ((msg_base_address+offset)+(src0.addr_size[n])).data_size[v] = src1[n].data_size[v];
            } else {
                ((msg_base_address+offset)+(src0.addr_size[n])).data_size[v] = src1[v].data_size[n];
            }
        }
    }
}
```

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>30:29</td><td><b>Address Type</b><table><tr><td>Format:</td><td>DP_ADDR_SURFACE_TYPE</td></tr></table>Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc.<table><tr><th>Restriction</th></tr><tr><td>Stateful store messages to UGM (SFID_F) is only allowed for SURFTYPE_BUFFER and SURFTYPE_SCRATCH. Stateful store messages to UGML (SFID_1) is only allowed for SURFTYPE_BUFFER. Store messages to SLM (SFID_E) and URB (SFID_6)must have DP_ADDR_TYPE as FLAT.</td></tr></table></td></tr>
</table>

DP_STORE - Store

28:25 Src0 Length

Format:	DP_ADDR_REG_SIZE

Specifies the size of the address payload, in registers.

Programming Notes
src0_length = roundup((addr_size * simd_size) / grf_size)) simd_size is 16 if transpose is 0. simd_size is 1 if transpose is 1.
src0_length = roundup((addr_size * simd_size) / grf_size)) simd_size is 8 or 16 if transpose is 0. simd_size is 1 if transpose is 1.

24:20 Dest Length

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description
0		No data returned in registers.

19:17 Cache

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

16 Reserved

Access:	RO
Format:	MBZ

15 Transpose

Format:	DP_TRANSPOSE

Specifies if the registers are a transposed data vector.

Restriction
Transposed vectors are restricted to Exec_Mask == 1.

14:12 Vector Size

Format:	DP_VECT_SIZE

Specifies the vector length of each data payload item.

Restriction	Source
Stores with vector size of 8 or more is restricted to EXEC_MASK <= 8 (lower 8 lanes).Address payload format in this case is either A16_PAYLOAD_SIMT8, A32_PAYLOAD_SIMT8 or A64_PAYLOAD_SIMT8, depending on DP_ADDR_SIZE.	
Stores with data size of d64 and vector size of 3 or more is restricted to EXEC_MASK <= 8 (lower 8 lanes). Address payload format in this case is either A16_PAYLOAD_SIMT8, A32_PAYLOAD_SIMT8 or A64_PAYLOAD_SIMT8, depending on DP_ADDR_SIZE.	
For dataports UGM, SLM and URB, if DP_TRANSPOSE is Off, maximum vector size supported is 8 for D32 and 4 for D64.	SFID_6, SFID_E, SFID_F

DP_STORE - Store

<table>
<tr><td></td><td></td><td><table><tr><td>For dataport UGML (SFID_1), if DP_TRANSPOSE is Off, maximum vector size supported is 4. Moreover, vector size of 3 is not supported.</td><td>SFID_1</td></tr></table></td></tr>
<tr><td></td><td>11:9</td><td><b>Data Size</b><table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>Specifies both bit size of the data payload item in memory and the bit size used in the register payload.<table><tr><th>Restriction</th><th>Source</th></tr><tr><td>8b and 16b data sizes are only supported with vector size 1 and Transpose off.</td><td></td></tr><tr><td>For UGML, data size of D64 is only allowed when Address size is A64. Also, when data size is D64, per-lane addresses must be QW aligned.</td><td>SFID_1</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b><table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>Specifies the bit size of each address payload item.<table><tr><th>Restriction</th></tr><tr><td>If DP_VECT_SIZE is 1, the addresses can be byte aligned. If DP_VECT_SIZE is greater than 1, addresses must be aligned to data size.</td></tr></table></td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Store Operation</b><table><tr><td>Default Value:</td><td>4 Store</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Store Cmask

DP_STORE_CMASK - Store Cmask

Source:	SFID_1, SFID_6, SFID_D, SFID_E, SFID_F
Length Bias:	1

Store untyped data to memory. For each enabled SIMT lane and enabled component mask, a scalar is written into memory from registers.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size. Cmask specifies how many sequential copies of the data payload are in the message.

Restriction	Source
This message is not supported for SFID_D (TGM).	
Store_cmask is not supported by data port URB.	SFID_6

Syntax

```
[(pred)] STORE_CMASK.sfid[.cache] (exec_mask) <addr_type[+offset]>src0_reg:addr_size
src1_reg:data_size[.cmask]
```

Pseudocode

```
msg_base_address = (surf_type == 'Scratch') ? Base + PThreadID*Pitch : Base
for (n = 0; n < 32; n++) {
    if (Msg.ChEn[n]) {
        for (m = v = 0; v < 4; v++) {
            if (cmask[v]) {
                ((msg_base_address+offset)+(src0.addr_size[n])).data_size[v] = src1[m].data_size[n];
                m++;
            }
        }
    }
}
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** Stateful store_cmask messages to UGM (SFID_F) is only allowed for SURFTYPE_BUFFER, SURFTYPE_SCRATCH and SURFTYPE_NULL. Stateful store_cmask messages to UGML (SFID_1) is only allowed for SURFTYPE_BUFFER. Store_mask messages to SLM (SFID_E) must have DP_ADDR_TYPE as FLAT.
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

DP_STORE_CMASK - Store Cmask

Programming Notes
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 16
src0_length = roundup((addr_size * num_coordinates * simd_size) / grf_size)) num_coordinates is the number of address coordinates used in message. For SLM, UGM and UGML it is always 1. For TGM it is between 1-4 (U, UV, UVR, UVRLOD). simd_size is 8 or 16

24:20 **Dest Length**

Format:	U5

Specifies the size of destination data register payload.

Value	Name	Description
0		No data returned in registers.

19:17 **Cache**

Format:	DP_CACHE_STORE

Specifies how the instruction overrides the cache settings.

16 **Reserved**

Access:	RO
Format:	MBZ

15:12 **Component Mask**

Format:	DP_CMASK

Specifies the component mask of each data payload item.

11:9 **Data Size**

Format:	DP_DATA_SIZE

Specifies both bit size of the data payload item in memory and the bit size used in the register payload.

Value	Name	Description
2		D32

8:7 **Address Size**

Format:	DP_ADDR_SIZE

Specifies the bit size of each address payload item.

6 **Reserved**

Access:	RO
Format:	MBZ

5:0 **Store Operation**

Default Value:	6 Store Cmask
Format:	Opcode

Store Uncompressed

DP_STORE_UNCOMPRESSED - Store Uncompressed

Source:	SFID_F
Length Bias:	1

Store untyped data to memory in uncompressed form. For each enabled SIMT lane, a vector is written into memory from registers.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size. If not transposed, Vector Size specifies how many sequential copies of the data payload are in the message. If transposed, the Exec_Mask specifies how many sequential copies of the data payload are in the message.

Restriction

Syntax

```
[(pred)] STORE_UNCOMPRESSED.sfid[.cache] (exec_mask)
<addr_type[+offset]>src0_reg:addr_size src1_reg:data_size[.vect_size][transpose]
```

Pseudocode

```
for (n = 0; n < 32; n++) {
    if (Msg.ChEn[n]) {
        for (v = 0; v < vect_size; v++) {
            if (transpose) {
                ((Base+offset)+(src0.addr_size[n])).data_size[v] = src1[n].data_size[v];
            } else {
                ((Base+offset)+(src0.addr_size[n])).data_size[v] = src1[v].data_size[n];
            }
        }
    }
}
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** This message is not allowed with DP_ADDR_TYPE as FLAT or BTI==255. This message is only allowed on UGM (SFID_F) and the surface must be SURFTYPE_BUFFER or NULL. The surface accessed by this message must be set as "3D compressible".
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.

<table>
<tr><th colspan="3">DP_STORE_UNCOMPRESSED - Store Uncompressed</th></tr>
<tr><td></td><td>24:20</td><td><b>Dest Length</b><table><tr><td>Format:</td><td>U5</td></tr></table>Specifies the size of destination data register payload.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td></td><td>No data returned in registers.</td></tr></table></td></tr>
<tr><td></td><td>19:17</td><td><b>Cache</b><table><tr><td>Format:</td><td>DP_CACHE_STORE</td></tr></table>Specifies how the instruction overrides the cache settings.<table><tr><th>Programming Notes</th></tr><tr><td>Store_uncompressed messages are always forced to "un-cacheable" in the L1 cache.</td></tr></table></td></tr>
<tr><td></td><td>16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15</td><td><b>Transpose</b><table><tr><td>Format:</td><td>DP_TRANSPOSE</td></tr></table>Specifies if the registers are a transposed data vector.<table><tr><th>Restriction</th></tr><tr><td>Transposed vectors are restricted to Exec_Mask == 1 and Vector_size greater than 1.</td></tr></table></td></tr>
<tr><td></td><td>14:12</td><td><b>Vector Size</b><table><tr><td>Format:</td><td>DP_VECT_SIZE</td></tr></table>Specifies the vector length of each data payload item.<table><tr><th>Restriction</th></tr><tr><td>StoreUncompressed with vector size of 8 or more is restricted to EXEC_MASK <= 16 (lower 16 lanes).</td></tr><tr><td>StoreUncompressed with data size of d64 and vector size of 3 or more is restricted to EXEC_MASK <= 16 (lower 16 lanes).</td></tr><tr><td>If DP_TRANSPOSE is Off, maximum vector size supported is 8 for D32 and 4 for D64.</td></tr></table></td></tr>
<tr><td></td><td>11:9</td><td><b>Data Size</b><table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>Specifies both bit size of the data payload item in memory and the bit size used in the register payload.<table><tr><th>Restriction</th></tr><tr><td>8b and 16b data sizes are only supported with vector size 1 and Transpose off.</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b><table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>Specifies the bit size of each address payload item.</td></tr>
</table>

DP_STORE_UNCOMPRESSED - Store Uncompressed

<table>
<tr><td></td><td></td><td><table><tr><th>Restriction</th></tr><tr><td>If DP_VECT_SIZE is 1, the addresses can be byte aligned. If DP_VECT_SIZE is greater than 1, addresses must be aligned to data size.</td></tr><tr><td>Only A32 is allowed.</td></tr></table></td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Store Operation</b><table><tr><td>Default Value:</td><td>28 Store Uncompressed</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Store Uncompressed Cmask

DP_STORE_UC_CMASK - Store Uncompressed Cmask

Source:	SFID_D
Length Bias:	1

Store untyped data to memory in uncompressed form. For each enabled SIMT lane and enabled component mask, a scalar is written into memory from registers.

Programming Notes
The src0 address payload format is selected by Address Size.
The src1 data payload format is selected by Data Size. Cmask specifies how many sequential copies of the data payload are in the message.

Syntax

```
[(pred)] STORE_UC_CMASK.sfid[.cache] (exec_mask) <addr_type[+offset]>src0_reg:addr_size
src1_reg:data_size[.cmask]
```

Pseudocode

```
for (n = 0; n < 32; n++) {
    if (Msg.ChEn[n]) {
        for (m = v = 0; v < 4; v++) {
            if (cmask[v]) {
                ((Base+offset)+(src0.addr_size[n])).data_size[v] = src1[m].data_size[n];
                m++;
            }
        }
    }
}
```

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30:29	**Address Type** Format: DP_ADDR_SURFACE_TYPE Specifies the format of the Extended Descriptor used with the address payload. Extracts the Base address and the global Offset used in address calculations from ExtDesc. **Restriction** This message is not allowed with DP_ADDR_TYPE as FLAT or BTI==255. This message is only allowed on TGM (SFID_D). The surface accessed by this message must be set as "3D compressible".
	28:25	**Src0 Length** Format: DP_ADDR_REG_SIZE Specifies the size of the address payload, in registers.
	24:20	**Dest Length** Format: U5 Specifies the size of destination data register payload. **Value** | **Name** | **Description** 0 | | No data returned in registers.

<table>
<tr><th colspan="3">DP_STORE_UC_CMASK - Store Uncompressed Cmask</th></tr>
<tr><td></td><td>19:17</td><td><b>Cache</b><table><tr><td>Format:</td><td>DP_CACHE_STORE</td></tr></table>Specifies how the instruction overrides the cache settings.</td></tr>
<tr><td></td><td>16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Component Mask</b><table><tr><td>Format:</td><td>DP_CMASK</td></tr></table>Specifies the component mask of each data payload item.</td></tr>
<tr><td></td><td>11:9</td><td><b>Data Size</b><table><tr><td>Format:</td><td>DP_DATA_SIZE</td></tr></table>Specifies both bit size of the data payload item in memory and the bit size used in the register payload.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>2</td><td></td><td>D32</td></tr></table></td></tr>
<tr><td></td><td>8:7</td><td><b>Address Size</b><table><tr><td>Format:</td><td>DP_ADDR_SIZE</td></tr></table>Specifies the bit size of each address payload item.<table><tr><th>Restriction</th></tr><tr><td>Must be A32.</td></tr></table></td></tr>
<tr><td></td><td>6</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>5:0</td><td><b>Store Operation</b><table><tr><td>Default Value:</td><td>32 Store Uncompressed Cmask</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
</table>

Subtraction with Borrow

subb - Subtraction with Borrow

Source:	Eulsa
Length Bias:	4
Predication:	true
Conditional Modifier:	true
Saturation:	true
Source Modifier:	false

The subb instruction performs component-wise subtraction of src0 and src1 and stores the results in dst, it also stores the borrow into acc. If the operation produces a borrow (src0 < src1), write 0x00000001 to acc, else write 0x00000000 to acc.

Format:

```
        [(pred)] subb[.cmod] (exec_size) dst src0 src1
```

Programming Notes

The accumulator is an implicit destination and thus cannot be an explicit destination operand.

Restriction

AccWrEn is required.

Syntax

```
[(pred)] subb[.cmod] (exec_size) reg reg reg
 [(pred)] subb[.cmod] (exec_size) reg reg imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < exec_size; n++ ) {
     if ( WrEn.chan[n] ) {
         dst.chan[n] = src0.chan[n] - src1.chan[n];
         acc.chan[n] = borrow(src.chan[n] - src1.chan[n]);
     }
 }
```

Src Types	Dst Types
UD	UD

DWord	Bit	Description
0..3	127:126	**Reserved** Exists If: ([Src1.IsImm]==false) Format: MBZ
	127:96	**Src1.ImmValue[31:0]** Exists If: ([Src1.IsImm]==true)

subb - Subtraction with Borrow

	125:122	**Reserved**		
			Exists If:	([Src1.IsImm]==false)
			Format:	MBZ
	121:120	**Src1.Mod**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**SrcMod**
	119:116	**Src1.VertStride**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**VertStride**
	115:113	**Src1.Width**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**Width**
	112	**Src1.AddrMode**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**AddrMode**
	111:98	**Src1.Operand**		
			Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Indirect)
			Format:	**IndirectOperand**
	111:98	**Src1.Operand**		
			Exists If:	([Src1.IsImm]==false) AND ([Src1.AddrMode]==Direct)
			Format:	**DirectOperand**
	97:96	**Src1.HorzStride**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**HorzStride**
	95:92	**CondCtrl**		
			Format:	**FlagModifier**
	91:88	**Src1.DataType**		
			Exists If:	([Src1.IsImm]==true)
			Format:	**ImmDataType**
	91:88	**Src1.DataType**		
			Exists If:	([Src1.IsImm]==false)
			Format:	**RegDataType**
	87:84	**Src0.VertStride**		
			Format:	**VertStride**
	83:81	**Src0.Width**		
			Format:	**Width**

<table>
<tr><th colspan="3">subb - Subtraction with Borrow</th></tr>
<tr><td></td><td>80</td><td><b>Src0.AddrMode</b><table><tr><td>Format:</td><td><b>AddrMode</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Src0.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Direct)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Exists If:</td><td>([Dst.AddrMode]==Indirect)</td></tr><tr><td>Format:</td><td><b>IndirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:48</td><td><b>Dst.HorzStride</b><table><tr><td>Format:</td><td><b>HorzStride</b></td></tr></table></td></tr>
<tr><td></td><td>47</td><td><b>Src1.IsImm</b>
This field indicate that Source 1 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false <b>[Default]</b></td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:44</td><td><b>Src0.Mod</b><table><tr><td>Format:</td><td><b>SrcMod</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>RegDataType</b></td></tr></table></td></tr>
<tr><td></td><td>43:40</td><td><b>Src0.DataType</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td><b>ImmDataType</b></td></tr></table></td></tr>
</table>

subb - Subtraction with Borrow

39:36 **Dst.DataType**

Format:	**RegDataType**

35 **Dst.AddrMode**

Format:	**AddrMode**

34 **Saturate**

Format:	**Saturate**

33 **AccWrCtrl**

Format:	**AccWrCtrl**

32 **AtomicCtrl**

Format:	**AtomicCtrl**

31 **MaskCtrl**

Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".

Value	Name	Description
0	Normal **[Default]**	Normal. Per channel write enable used for final write enable generation.
1	NoMask	NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.

30 **Reserved**

29 **CmptCtrl**

Format:	MBZ

Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.

Value	Name	Description
0	NoCompaction **[Default]**	No compaction. 128-bit native instruction supporting all instruction options.
1	Compacted	Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.

28 **PredInv**

This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields

subb - Subtraction with Borrow

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr>
<tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr>
</table>
</td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b>
<table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>
This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
<tr><td></td><td>21:19</td><td><b>ChanOff</b>
<table><tr><td>Format:</td><td><b>ChanOff</b></td></tr></table>
This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.</td></tr>
<tr><td></td><td>18:16</td><td><b>ExecSize</b>
<table><tr><td>Format:</td><td><b>ExecSize</b></td></tr></table>
This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.</td></tr>
<tr><td></td><td>15:0</td><td><b>Header</b>
<table><tr><td>Format:</td><td><b>Header</b></td></tr></table>
</td></tr>
</table>

Synchronize

sync - Synchronize

Source:	EuIsa
Length Bias:	4
Predication:	false
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false
Subfunctions:	SyncFC[95:92]

Wait on Dependency performs various operations related to synchronization such as waiting on registers (barriers registers) or for software scoreboarding (SWSB), which is used to specify pipeline hazards to the EU. The instruction has several sub-operations (function controls), including:

- **nop** (0000b): no operation (encoded SWSB information available to every instruction is still honored). This might be used if an instruction depends on two different out-of-order sources. The consumer can only specify a dependency on one, hence an extra instruction must be added for this.
- Reserved (0001b): reserved for future expansion.
- **allrd** (0010b): blocks until all out-of-order sources are read (e.g. input arguments to a send or math op).
- **allwr** (0011b): blocks until all out-of-order destinations are written back (e.g. writes from a send or math op).
- Reserved (0100-1100b): reserved for future expansion
- **fence** (1101b): blocks on the notification register for fence response. When fence response is received from message gateway, bit 0 of n0.2 notification register is set. Instruction sync.fence blocks until the bit is set and clears before progressing to the next instruction.
- **bar** (1110b): blocks on the notification register for barriers response. When barrier response is received from message gatewaybits corresponding to the barrier id are set in the notification register n0. Instruction sync.bar(barrier id) blocks until the bit corresponding to the barrier id is set, and clears it before progressing to the next instruction.
- **host** (1111b): blocks on the notification register for host interaction. When host notification is received, the bit 0 of n1 notification register is set. Instruction sync.host blocks until the bit is set and clears it before progressing to the next instruction.

See the SyncFC BXML enum for more information.

Format: sync.[sync_fc] src0

Programming Notes

The format is that of a basic instruction. The immediate operand is encoded as src0 and may explicitly be null if not used. Src1 and dst must be null.

Syntax

sync.nop null [instopts]
sync.allrd (null | imm32) [instopts]

sync - Synchronize

sync.allwr (null | imm32) [instopts] sync.bar null[instopts] sync.host null [instopts]

Pseudocode

```
     Evaluate(WrEn);
          switch (func) {
          case nop:
             // regular SWSB dep check from instruction options executes
             break;
          case allrd:
             for (sbid = 0; sbid < MAX_SBIDS; sbid++) {
                if (Src0.IsImm) {
                  // wait until selected OOO reads are finished
                  if(Src0.ImmValue[sbid]) wait_on_sbid_read_access(sbid); // transition to
wait_dst or idle
               } else {
                   // wait until all OOO reads are finished
                   wait_on_sbid_read_access(sbid); // transition to wait_dst or idle
                }
             }
             break;
          case allwr:
             for (sbid = 0; sbid < MAX_SBIDS; sbid++) {
               if (Src0.IsImm) {
                  // wait until selected OOO writes are finished
                   if(Src0.ImmValue[sbid]) wait_on_sbid_write_access(sbid); // transition
to idle
               } else {
                 // wait until selected OOO writes are finished
                   wait_on_sbid_write_access(sbid); // transition to idle
               }
             }
             break;
          case bar:
             wait_on_barrier_notification(1 « Src0);
             if (Src0.IsImm) {
                wait_on_barrier_notification(1 « Src0.ImmValue[4:0]) // waits until the
corresponding barrier bit is set
             else if (Src0.RegFile == ARF) {
                wait_on_barrier_notification(1 « Src0[4:0]) // waits until the
corresponding barrier bit is set
             else {
                wait_on_barrier_notification(1) // waits until the barrier bit 0 is set
             }
             break;
          case host:
             wait_on_host_notification(); // waits until the host signals the host barrier
             break;
          }
```

Src Types
*B,*W,*D,*Q, HF, F, DF

DWord	Bit	Description

sync - Synchronize

0..3	127:96	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	127:96	**Src0.ImmValue[31:0]** Exists If: ([Src0.IsImm]==true)
	95:92	**SyncCtrl** Format: SyncFC
	91:88	**Reserved** Format: MBZ
	87	**Reserved** Format: MBZ
	86:80	**Reserved** Format: MBZ
	79:66	**Reserved** Format: MBZ
	65:50	**Reserved** Format: MBZ
	49:48	**Dst.HorzStride** **Value** — **Name** 01b — 1 elements **[Default]** Others — Reserved
	47	**Reserved** Format: MBZ
	46	**Src0.IsImm** This field indicate that Source 0 operand is carrying an immediate value. **Value** — **Name** 0 — false 1 — true
	45:44	**Reserved** Format: MBZ
	43:40	**Src0.DataType** Exists If: ([Src0.IsImm]==true) Format: ImmDataType
	43:40	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ

sync - Synchronize

Bits	Description
39:33	**Reserved**

Format:	MBZ

Bits	Description
32	**AtomicCtrl**

Format:	AtomicCtrl

Bits	Description
31	**MaskCtrl** Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".

Value	Name	Description
0	Normal **[Default]**	Normal. Per channel write enable used for final write enable generation.
1	NoMask	NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.

Bits	Description
30	**Reserved**
29	**CmptCtrl**

Format:	MBZ

Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.

Value	Name	Description
0	NoCompaction **[Default]**	No compaction. 128-bit native instruction supporting all instruction options.
1	Compacted	Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.

Bits	Description
28	**PredInv** This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields

Value	Name	Description
0	Positive **[Default]**	Positive polarity of predication. Use the predication mask produced by PredCtrl.
1	Negative	Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.

sync - Synchronize

	27:24	**PredCtrl** Format: PredCtrl This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.
	23	**FlagRegNum[0]** This field specifies bit[0] of the register number for a flag register operand.
	22	**FlagSubRegNum** This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.
	21:19	**ChanOff** Format: ChanOff This field provides offset information for ARF selection. This can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: ExecSize This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: Header

Trace Ray Message Descriptor

TRACERAY_MSD - Trace Ray Message Descriptor

Source:	SFID_RTA
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15
	24:20	**Reserved** Access: RO Format: MBZ
	19	**header Present** Format: Enable **Programming Notes** Must be programmed to 0
	18	**Reserved** Access: RO Format: MBZ
	17:14	**Message Type** Format: Opcode Trace Ray Message to the Ray Tracing HW Acceleration Shared Function. **Value** / **Name** 00h /
	13:9	**Reserved** Access: RO Format: MBZ
	8	**SIMD mode** Format: MDC_SM2 Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Reserved** Access: RO Format: MBZ

Typed Surface CCS Operation MSD

MSD_TS_CCS_OP - Typed Surface CCS Operation MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 0Ch Format: Opcode Typed Surface CCS update message

MSD_TS_CCS_OP - Typed Surface CCS Operation MSD		
	13:12	**Slot Group** Format: **MDC_SG3** Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**CCS Operation** Format: **MDC_CCS_SEC_OP** Specifies which CCS operation is performed.
	7:0	**Binding Table Index** Format: **MDC_BTS** Specifies the Binding Table Index for the message

Typed Surface Read MSD

MSD1R_TS - Typed Surface Read MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 05h Format: Opcode Typed Surface Read message

MSD1R_TS - Typed Surface Read MSD

	13:12	**Slot Group** Format: **MDC_SG3** Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Channel Mask** Format: **MDC_CMASK** Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: **MDC_BTS** Specifies the Binding Table Index for the message

Typed Surface Uncompressed Write MSD

MSD_TS_UCW - Typed Surface Uncompressed Write MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 0Dh Format: Opcode Typed Surface Uncompressed Write message

MSD_TS_UCW - Typed Surface Uncompressed Write MSD

	13:12	**Slot Group** Format: **MDC_SG3** Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Channel Mask** Format: **MDC_CMASK** Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: **MDC_BTS** Specifies the Binding Table Index for the message

Typed Surface Write MSD

MSD1W_TS - Typed Surface Write MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 0Dh Format: Opcode Typed Surface Write message

MSD1W_TS - Typed Surface Write MSD

	13:12	**Slot Group** Format: **MDC_SG3** Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Channel Mask** Format: **MDC_CMASK** Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: **MDC_BTS** Specifies the Binding Table Index for the message

Untyped Surface CCS Operation MSD

MSD_US_CCS_OP - Untyped Surface CCS Operation MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 08h Format: Opcode Untyped Surface CCS update message.

MSD_US_CCS_OP - Untyped Surface CCS Operation MSD

	13:12	**SIMD Mode** Format: MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**CCS Operation** Format: MDC_CCS_SEC_OP Specifies which CCS Operation is performed.
	7:0	**Binding Table Index** Format: MDC_BTS_A32 Specifies the Binding Table Index for the message

Untyped Surface Read MSD

MSD1R_US - Untyped Surface Read MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 01h Format: Opcode Untyped Surface Read message

MSD1R_US - Untyped Surface Read MSD

	13:12	**SIMD Mode** Format: | MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: | MDC_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: | MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Untyped Surface Uncompressed Write MSD

MSD_US_UCW - Untyped Surface Uncompressed Write MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 09h Format: Opcode Untyped Surface Uncompressed Write message

MSD_US_UCW - Untyped Surface Uncompressed Write MSD

	13:12	**SIMD Mode** Format: **MDC_SM3** Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: **MDC_UW_CMASK** Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: **MDC_BTS_A32** Specifies the Binding Table Index for the message

Untyped Surface Write MSD

MSD1W_US - Untyped Surface Write MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 09h Format: Opcode Untyped Surface Write message

MSD1W_US - Untyped Surface Write MSD

	13:12	**SIMD Mode** Format: | MDC_SM3 Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Format: | MDC_UW_CMASK Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Format: | MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

URB Dword Read MSD

MSDUR_DWS - URB Dword Read MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Default Value: 1 Present Format: Opcode Indicates that the message requires a header.
	18	**Reserved** Access: RO Format: MBZ
	17	**Per Slot Offset Present** Format: Enable Specifies if per-slot offset message payload is present.
	16	**Reserved** Access: RO Format: MBZ
	15	**Channel Mask Present** Default Value: 0 Not Present Format: Opcode Must be clear on read messages, indicating the Channel Mask Message phase is not present.
	14:4	**Global Offset** Format: U11 Specifies the offset, in units of Oword elements, from the start of the URB handle for the access. If **Per Slot Offset Present** is set, this global offset is added to each of the slot offsets to form the overall offset.

MSDUR_DWS - URB Dword Read MSD

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>[0-2047]</td><td></td></tr>
</table>
</td></tr>
<tr><td></td><td>3:0</td><td>
<b>URB Opcode</b>
<table>
<tr><td>Format:</td><td>Opcode</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>8</td><td>URB_SIMD8_READ <b>[Default]</b></td><td>SIMD8 URB Dword Read message. Reads 1..8 Dwords, based on RLEN.</td></tr>
</table>
</td></tr>
</table>

URB Dword Write MSD

MSDUW_DWS - URB Dword Write MSD

Source: EuSubFunctionReadOnlyDataPort

Length Bias: 1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Default Value: 1 Present Format: Opcode Indicates that the message requires a header.
	18	**Reserved** Access: RO Format: MBZ
	17	**Per Slot Offset Present** Format: Enable Specifies if per-slot offset message payload is present. If present, it will be added to the **Global Offset**.
	16	**Reserved** Access: RO Format: MBZ
	15	**Channel Mask Present** Default Value: 0 Not Present Format: Opcode Indicates the channel Mask Message phase is not present.
	14:4	**Global Offset** Format: U11 Specifies the offset, in units of Oword elements, from the start of the URB handle for the access. If **Per Slot Offset** is set, the global offset is added to those offsets to form the overall offset.

MSDUW_DWS - URB Dword Write MSD

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>[0-2047]</td><td></td></tr>
</table>
</td></tr>
<tr><td></td><td>3:0</td><td>
<b>URB Opcode</b>
<table>
<tr><td>Format:</td><td>Opcode</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>7</td><td>URB_SIMD8_WRITE <b>[Default]</b></td><td>SIMD8 URB Dword Write message. Writes1..8 Dwords, based on RLEN and Channel Mask.</td></tr>
</table>
</td></tr>
</table>

URB Fence

MSD_URBFENCE - URB Fence

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Default Value: 1 Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).
	24:20	**Response Length** Default Value: 1 Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload.
	19:4	**Reserved** Access: RO Format: MBZ
	3:0	**URB Opcode** Format: Opcode

Value	Name	Description
9	URB_FENCE **[Default]**	URB Fence message

URB Masked Dword Write MSD

MSDUW_MDWS - URB Masked Dword Write MSD

Source:	EuSubFunctionReadOnlyDataPort
Length Bias:	1

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Default Value: 1 Present Format: Opcode Indicates that the message requires a header.
	18	**Reserved** Access: RO Format: MBZ
	17	**Per Slot Offset Present** Format: Enable Specifies if per-slot offset message payload is present. If present, it will be added to the **Global Offset**.
	16	**Reserved** Access: RO Format: MBZ
	15	**Channel Mask Present** Default Value: 1 Present Format: Opcode Indicates the Channel Mask Message phase is present and will be used to mask which data elements written.
	14:4	**Global Offset** Format: U11 Specifies the offset, in units of Oword elements, from the start of the URB handle for the access. If **Per Slot Offset** is set, the global offset is added to those offsets to form the overall offset.

<table>
<tr><th colspan="3">MSDUW_MDWS - URB Masked Dword Write MSD</th></tr>
<tr><td></td><td></td><td><table><tr><th>Value</th><th>Name</th></tr><tr><td>[0-2047]</td><td></td></tr></table></td></tr>
<tr><td></td><td>3:0</td><td><b>URB Opcode</b><table><tr><td>Format:</td><td>Opcode</td></tr></table><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>7</td><td>URB_SIMD8_WRITE <b>[Default]</b></td><td>SIMD8 URB Dword Write message. Writes1..8 Dwords, based on RLEN and Channel Mask.</td></tr></table></td></tr>
</table>

VD_CONTROL_STATE

VD_CONTROL_STATE

Source:	VideoCS
Length Bias:	2

This command can be used in HCPpipe.
For HCP, it is selected with the **Media Instruction Opcode "7h".**
Each command has assigned a media instruction command as defined in DWord 0, BitField 22:16. It will be different between HCP.

This command is used to modify the control of HCP pipe. It can be inserted anywhere within a frame. It can be inserted multiple times within a frame as well.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name for HCP Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: Ah VD_CONTROL_STATE Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 2h — Name:
1..2	63:0	**VD Control State Body** Format: VD_CONTROL_STATE_BODY

VD_PIPELINE_FLUSH

VD_PIPELINE_FLUSH

Source:	VideoCS
Length Bias:	2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:23	**Media Command Opcode** Default Value: Fh Extended command Format: OpCode
	22:21	**SubOpcodeA** Default Value: 0h Format: OpCode
	20:16	**SubOpcodeB** Default Value: 0h Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWORD_COUNT_n** Default Value: 0h Excludes DWord (0) Format: =n Total Length - 2
1	31:23	**Reserved** Access: RO Format: MBZ
	22	**Reserved** Access: RO Format: MBZ
	21	**AVP pipeline command flush** Format: U1

VD_PIPELINE_FLUSH

	20	**HuC Pipeline command flush**	Format:	U1
	19	**MFX pipeline command flush**	Format:	U1
	18	**Reserved**		
	17	**VD-ENC pipeline command flush**	Format:	U1
	16	**HEVC pipeline command flush**	Format:	U1
	15:8	**Reserved**	Access: Format:	RO MBZ
	7	**Reserved**	Access: Format:	RO MBZ
	6	**AVP pipeline Done**	Format:	U1
	5	**HuC pipeline Done**	Format:	U1
	4	**VD command/message parser Done**	Format:	U1
	3	**MFX pipeline Done**	Format:	U1
	2	**Reserved**		
	1	**VD-ENC pipeline Done**	Format:	U1
	0	**HEVC pipeline Done**	Format:	U1

VEBOX_STATE

<table>
<tr><th colspan="3">VEBOX_STATE</th></tr>
<tr><td colspan="3">Source: VideoEnhancementCS
Length Bias: 2</td></tr>
<tr><td colspan="3">This command controls the internal functions of the VEBOX. This command has a set of indirect state buffers:
• DN/DI state
• IECP general state
• IECP Gamut Expansion/Compression state
• IECP Gamut Vertex Table state
• Capture Pipe state</td></tr>
<tr><td colspan="3">Adds the LACE LUT Table as an indirect state buffer.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline</b><table><tr><td>Default Value:</td><td>2h Media</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><b>Command OpCode</b><table><tr><td>Default Value:</td><td>4h VEBOX</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>23:21</td><td><b>SubOpcode A</b><table><tr><td>Default Value:</td><td>0h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>20:16</td><td><b>SubOpcode B</b><table><tr><td>Default Value:</td><td>2h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b><table><tr><td>Format:</td><td>=n</td></tr></table><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>11h</td><td></td><td>(Excludes DWords 0, 1)</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">VEBOX_STATE</th></tr>
<tr><td>1</td><td>31:25</td><td><b>State Surface Control Bits</b>
All Indirect state buffers use state surface control bits, only exception being 3D LUT state buffer for which the state surface control bits are tied to 0.
See definition under "VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS" Bits[6:0] is only used.</td></tr>
<tr><td></td><td>24</td><td><b>FP16 mode enable</b></td></tr>
<tr><td></td><td>23</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>22</td><td><b>Gamut Expansion Position</b>
If Gamut Expansion is enabled, it can be configured either in front or backend of the IECP pipe using this bit.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Gamut Expansion at the Backend of IECP pipe</td></tr>
<tr><td>1b</td><td>Gamut Expansion at the Front of IECP pipe</td></tr>
</table></td></tr>
<tr><td></td><td>21</td><td><b>Forward Gamma Correction Enable</b>
<table>
<tr><td>Format:</td><td>Enable</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Single Pipe IECP Enable must also be set if this is enabled.</td></tr>
<tr><td>When enabled the forward gamma will always be in front of the IECP pipe. In case disabled it will be always configured as Gamut expansion. Gamut Expansion, HDR and Forward Gamma Correction are mutually exclusive.</td></tr>
</table></td></tr>
<tr><td></td><td>20</td><td><b>Scalar Mode</b>
When Scalar Mode is enabled, all other VEBOX functions must be disabled (DN/DI/DM/IECP/Chroma upsampling).</td></tr>
<tr><td></td><td>19</td><td><b>Single Pipe Enable</b>
<table>
<tr><td>Indicates that the Capture Pipe features that only exist in a single pipe can be enabled.</td></tr>
<tr><td>This bit must be set if any of the following features are enabled: Demosaic Denoise with one of the RGBA input formats IECP only mode with Forward Gamma Correction enabled with RGB input formats (All other modes are not supported in single pipe)</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>1</td><td>Enable</td></tr>
<tr><td>0</td><td>Default <b>[Default]</b></td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Note that the pixel throughput is 1/2 when this mode is selected. The <b>Global IECP Enable</b> must also be set.</td></tr>
</table></td></tr>
<tr><td></td><td>18</td><td><b>Disable Temporal Denoise Filter</b>
If set this bit will force the denoise filter to only use the spatial filter. This will eliminate the read of the previous denoise surface and STMM/Denoise History surface and the write of the current denoised surface and STMM/Denoise History surface.</td></tr>
</table>

VEBOX_STATE

<table>
<tr><td></td><td></td><td><table><tr><th>Programming Notes</th></tr><tr><td><b>The Global IECP Enable</b> or <b>Demosaic Enable</b> must be set along with this bit. This bit must be set if the input to Denoise is RGB. This bit must not be set if the Deinterlacer is enabled. This bit must be clear if both <b>DN Enable</b>=0 and <b>Hot Pixel Filtering Enable</b>=0This bit must be set if <b>Hot Pixel Filtering Enable</b>=1 and both DN and DI are disabled</td></tr></table></td></tr>
<tr><td></td><td>17</td><td><b>Disable Encoder Statistics</b>
If set this bit will disable writing the per block Encoder statistics. The memory format is not changed, so the area set aside for these statistics will still be there.</td></tr>
<tr><td></td><td>16</td><td><b>LACE Correction Enable</b>
This bit enables the correction of the image according to the local ACE LUT tables. This is independent from the enable for the collection of LACE histograms.<table><tr><th>Programming Notes</th></tr><tr><td>LACE correction is only enabled if both this bit and the <b>Global IECP Enable</b> are set. The <b>ACE Enable</b> bit should also be set if this bit is set, since ACE correction can be used for part of the luma range instead of LACE.</td></tr></table></td></tr>
<tr><td></td><td>15:14</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>13</td><td><b>Hot Pixel Filtering Enable</b>
Enables hot pixel detection/filtering.</td></tr>
<tr><td></td><td>12</td><td><b>Alpha Plane Enable</b>
Enables the reading of an independent Alpha plane. Mutually exclusive with Vignette Enable. If <b>Alpha from State Select</b> is set it overrides this bit.<table><tr><th>Programming Notes</th></tr><tr><td>IECP must also be enabled and output format must have alpha if this bit is enabled. Should be 0 if Alpha from State Select is 1.</td></tr></table></td></tr>
<tr><td></td><td>11</td><td><b>Vignette Enable</b>
Enables Vignette Correction surface read and correction in IECP. Mutually exclusive with Alpha Plane Enable.<table><tr><th>Programming Notes</th></tr><tr><td>Demosaic must also be enabled if this bit is enabled.</td></tr></table></td></tr>
<tr><td></td><td>10</td><td><b>Demosaic Enable</b>
The Demosaic will be used, and White balance statistics will be gathered. The Capture Pipe State Table will be read. This bit is mutually exclusive with <b>DI Enable.</b><table><tr><th>Programming Notes</th></tr><tr><td>IECP must also be enabled if this bit is enabled.</td></tr></table></td></tr>
<tr><td></td><td>9:8</td><td><b>DI Output Frames</b>
Indicates which frames to output in DI mode.<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>Output Both Frames</td></tr></table></td></tr>
</table>

VEBOX_STATE

01b	Output Previous Frame Only
10b	Output Current Frame Only

Programming Notes
Field is ignored if DI Enable = 0.If Previous Frame Only or Current Frame Only are selected, then the **LACE Single Histogram Set** must not try to collect a histogram from the disabled frame.
Field must be programmed to 10 (Output Current Frame Only) for DI First Frame.

7:6 **Reserved**

Access:	RO
Format:	MBZ

5 **DN/DI First Frame**

Format:	Enable

Indicates that this is the first frame of the stream, so previous clean is not available.

Value	Name
0	Not first field; previous clean surface state is valid
1	First field; previous clean surface state is invalid

Programming Notes
If both DN and DI are disabled, this bit must be 0.

4 **DI Enable**

Format:	Enable

Deinterlacer is bypassed if this is disabled: the output is the same as the input (same as a 2:2 cadence). FMD and STMM are not calculated and the values in the response message are 0.

Value	Name
0	Do not calculate DI
1	Calculate DI

3 **DN Enable**

Format:	Enable

Denoise is bypassed if this is low - BNE is still calculated and output, but the denoised fields are not. VDI does not read in the denoised previous frame but uses the pointer for the original previous frame.

Value	Name
0	Do not denoise frame
1	Denoise frame

Programming Notes
If DN and/or Hotpixel are the only functions enabled then the only output is the Denoised Output which is the same surface format as the input. To get a format conversion with DN only, enable the Global IECP bit, but disable all the individual functions. The IECP output uses the

VEBOX_STATE

		output surface format. If DN is used with RGB then the **Global IECP** Enable must also be
	2	**Global IECP Enable** Indicates if any of the IECP features is enabled. If this is disabled then no state will be read from any of the state pointers. If set then the IECP state will be read.
	1	**Color Gamut Compression Enable** Indicates if the Gamut Compression feature is enabled. If set then the Gamut State will be read. VEB_VERTEXTABLE_STATE is only needed if this bit is set.
	0	**Color Gamut Expansion Enable** Indicates if the Gamut Expansion feature is enabled. If set then the Gamut State will be read. This can be enabled only if Single pipe enable is disabled.
2	31:12	**DN/DI State Pointer Low** Format: GraphicsAddress[31:12] Bits 31:12 of the starting address of the DN/DI State buffer. This points to a buffer containing the 10 Dwords of the DN/DI state. When Scalar mode is enabled this pointer is used for Scalar state table.
	11:0	**Reserved** Access: RO Format: MBZ
3	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**DN/DI State Pointer High** Format: GraphicsAddress[47:32] Bits 47:32 of the starting address of the DN/DI State Buffer. When Scalar mode is enabled this pointer is used for Scalar state table.
4	31:12	**IECP State Pointer Low** Format: GraphicsAddress[31:12] Bits 31:12 of the starting address of the IECP State buffer. This points to a buffer containing the 64 Dwords of IECP state.
	11:0	**Reserved** Access: RO Format: MBZ
5	31:16	**Reserved** Access: RO Format: MBZ

VEBOX_STATE

	15:0	**IECP State Pointer High** Format: GraphicsAddress[47:32] Bits 47:32 of the starting address of the IECP State Buffer Table.
6	31:12	**Gamut/HDR State Pointer Low** Format: GraphicsAddress[31:12] Bits 31:12 of the starting address of the State Buffer. If Gamut Expansion is enabled, this points to a buffer containing the Gamut Expansion Gamma Correction state. If HDR is enabled, this points to a buffer containing the HDR state.
	11:0	**Reserved** Access: RO Format: MBZ
7	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Gamut/HDR State Pointer High** Format: GraphicsAddress[47:32] Bits 47:32 of the starting address of the Gamut/HDR State Buffer.
8	31:12	**Vertex Table State Pointer Low** Format: GraphicsAddress[31:12] Bits 31:12 of the starting address of the Vertex Table. This points to a buffer containing the 512 Dwords of the Gamut Compression Vertex Table.
	11:0	**Reserved** Access: RO Format: MBZ
9	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Vertex Table State Pointer High** Format: GraphicsAddress[47:32] Bits 47:32 of the starting address of the Vertex State Buffer.
10	31:12	**Capture Pipe State Pointer Low** Format: GraphicsAddress[31:12] Bits 31:12 of the starting address of the Capture Pipe State Table. This points to a buffer containing the X Dwords of the Capture Pipe State.
	11:0	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">VEBOX_STATE</th></tr>
<tr><td>11</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>Capture Pipe State Pointer High</b><table><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table>Bits 47:32 of the starting address of the Capture Pipe State Table.</td></tr>
<tr><td>12</td><td>31:12</td><td><b>LACE LUT Table State Pointer Low</b><table><tr><td>Format:</td><td>GraphicsAddress[31:12]</td></tr></table>Bits [31:12] of the starting address of the LACE Look-up Tables.</td></tr>
<tr><td></td><td>11:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>13</td><td>31:30</td><td><b>Arbitration Priority Control - For LACE LUT</b><table><tr><td>Format:</td><td>U2</td></tr></table>This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Highest priority</td></tr><tr><td>1</td><td>Second highest priority</td></tr><tr><td>2</td><td>Third highest priority</td></tr><tr><td>3</td><td>Lowest priority</td></tr></table></td></tr>
<tr><td></td><td>29:16</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:0</td><td><b>LACE LUT Table State Pointer High</b><table><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table>Bits [47:32] of the starting address of the LACE Look-up Tables.</td></tr>
<tr><td>14..15</td><td>63:12</td><td><b>Gamma Correction Values Address</b><table><tr><td>Format:</td><td>VIRTUAL_ADDR[63:12]</td></tr></table>Specifies the 4K byte aligned address reading the Gamma Correction Values in case enabled.</td></tr>
<tr><td></td><td>11:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>16</td><td>31:12</td><td><b>3D LUT State Pointer Low</b><table><tr><td>Format:</td><td>GraphicsAddress[31:12]</td></tr></table>Bits [31:12] of the starting address of the 3D LUT.</td></tr>
</table>

<table>
<tr><th colspan="3">VEBOX_STATE</th></tr>
<tr><td></td><td>11:0</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>17</td><td>31:30</td><td><b>Arbitration Priority Control - For 3D LUT</b><table><tr><td>Format:</td><td>U2</td></tr></table>This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Highest Priority</td></tr><tr><td>1</td><td>Second highest priority</td></tr><tr><td>2</td><td>Third highest priority</td></tr><tr><td>3</td><td>Lowest priority</td></tr></table></td></tr>
<tr><td></td><td>29</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>28:24</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>23:22</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>21:16</td><td><b>3D LUT MOCS table</b>
These are surface control bits for VEBOX 3DLUT data requests to GAV</td></tr>
<tr><td></td><td>15:0</td><td><b>3D LUT State Pointer High</b><table><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table>Bits [47:32] of the starting address of the 3D LUT.</td></tr>
<tr><td>18</td><td>31</td><td><b>3D LUT Enable</b><table><tr><td>Default Value:</td><td>0</td></tr><tr><td>Format:</td><td>Enable</td></tr></table><b>3D LUT is required only if this is enabled.</b><table><tr><th>Programming Notes</th></tr><tr><td>Single Pipe IECP Enable must also be set if this is enabled.</td></tr></table><table><tr><th>Restriction</th></tr><tr><td>The frame height needs to be multiple of 8 when enabling 3dlut in VEBOX dual pipe mode.</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">VEBOX_STATE</th></tr>
<tr><td></td><td>30:29</td><td><b>3D LUT Size</b>
<table><tr><td>Format:</td><td>U2</td></tr></table>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>33x33x33</td></tr><tr><td>01b</td><td>17x17x17</td></tr><tr><td>10b</td><td>65x65x65</td></tr></table></td></tr>
<tr><td></td><td>28:23</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>22:16</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>15:14</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>13:12</td><td><b>Frame statistics ID</b>
<table><tr><td>Format:</td><td>U2</td></tr></table>
This field specifies the Statistics Surface ID number to the VEBOX to writeout the frame statistics.</td></tr>
<tr><td></td><td>11</td><td><b>Bypass Chroma Downsampling</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
<b>When enabled will drop chroma samples at odd position and not use the co-sited offsets.</b></td></tr>
<tr><td></td><td>10</td><td><b>Bypass Chroma Upsampling</b>
<table><tr><td>Format:</td><td>U1</td></tr></table>
<b>When enabled will replicate chroma samples at odd position and not use the co-sited offsets.</b></td></tr>
<tr><td></td><td>9:7</td><td><b>Chroma Downsampling Co-Sited Vertical Offset</b>
<table><tr><td>Format:</td><td>U3</td></tr></table>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td><b>[Default]</b></td></tr><tr><td>[0,2]</td><td>Valid Range</td></tr></table></td></tr>
<tr><td></td><td>6:5</td><td><b>Chroma Downsampling Co-Sited Horizontal Offset</b>
<table><tr><td>Format:</td><td>U2</td></tr></table>
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td><b>[Default]</b></td></tr><tr><td>[0,2]</td><td>Valid Range</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">VEBOX_STATE</th></tr>
<tr><td></td><td>4:2</td><td><b>Chroma Upsampling Co-Sited Vertical Offset</b>
<table><tr><td>Format:</td><td>U3</td></tr></table>
<table><tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td><b>[Default]</b></td></tr>
<tr><td>[0,4]</td><td>Valid Range</td></tr></table></td></tr>
<tr><td></td><td>1:0</td><td><b>Chroma Upsampling Co-Sited Horizontal Offset</b>
<table><tr><td>Format:</td><td>U2</td></tr></table>
<table><tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td><b>[Default]</b></td></tr>
<tr><td>[0,1]</td><td>Valid Range</td></tr></table></td></tr>
</table>

VEBOX_SURFACE_STATE

VEBOX_SURFACE_STATE

Source:	VideoEnhancementCS
Length Bias:	2

The input and output data containers accessed are called "surfaces". Surface state is sent to VEBOX via an inline state command rather than using binding tables. SURFACE_STATE contains the parameters defining each surface to be accessed, including its size, format, and offsets to its subsurfaces. The surface's base address is in the execution command. Despite having multiple input and output surfaces, we limit the number of surface states to one for input surfaces and one for output surfaces. The other surfaces are derived from the input/output surface states.

The Current Frame Input surface uses the Input SURFACE_STATE

The Previous Denoised Input surface uses the Input SURFACE_STATE.
(For 16-bit Bayer pattern inputs this will be 16-bit.)

The Current Denoised Output surface uses the Input SURFACE_STATE.
(For 16-bit Bayer pattern inputs this will be 16-bit.)

The STMM/Noise History Input surface uses the Input SURFACE_STATE with Tile-Y and Width/Height a multiple of 4.

The STMM/Noise History Output surface uses the Input SURFACE_STATE with Tile-Y and Width/Height a multiple of 4.

The Current Deinterlaced/IECP Frame Output surface uses the Output SURFACE_STATE.

The Previous Deinterlaced/IECP Frame Output surface uses the Output SURFACE_STATE.

The FMD per block output / per Frame Output surface uses the Linear SURFACE_STATE (see note below).

The Alpha surface uses the Linear A8 SURFACE_STATE with Width/Height equal to Input Surface. Pitch is width rounded to next 64.

The Skin Score surface uses the Output SURFACE_STATE.

The STMM height is the same as the Input Surface height except when the input **Surface Format** is Bayer Pattern and the **Bayer Pattern Offset** is 10 or 11, in which case the height is the input height + 4.
For Bayer pattern inputs when the **Bayer Pattern Offset** is 10 or 11, the Current Denoised Output/Previous Denoised Input will also have a height which is the input height + 4.For Bayer pattern inputs only the Current Denoised Output/Previous Denoised Input are in Tile-Y.

The linear surface for FMD statistics is linear (not tiled). The height of the per block statistics is (Input Height +3)/4 - the Input Surface height in pixels is rounded up to the next even 4 and divided by 4. The width of the per block section in bytes is equal to the width of the Input Surface in pixels rounded up to the next 16 bytes. The pitch of the per block section in bytes is equal to the width of the Input Surface in pixels rounded up to the next 64 bytes.

The STMM surfaces must be identical to the Input surface except for the tiling mode must be Tile-Y and the pitch is specified in DW7. The pitch for the Current Denoised Output/Previous Denoised Input is specified in DW7. The width and height must be a multiple of 4 rounded up from the input height.

The Vignette Correction surface uses the Linear 16-bit SURFACE_STATE with :
Width=(Ceil(Image Width / 4) +1) * 4
Height= Ceil(Image Height / 4) +1

VEBOX_SURFACE_STATE

Pitch in bytes is (vignette width *2) rounded to the next 64

Programming Notes
VEBOX may write to memory between the surface width and the surface pitch for output surfaces.
VEBOX can support a frame level X/Y offset which allows processing of 2 side-by-side frames for certain 3D video formats.
The X/Y Offset for Frame state applies only to the Current Frame Input and the Current Deinterlaced/IECP Frame Output and Previous Deinterlaced/IECP Frame Output. The statistics surfaces, the denoise feedback surfaces and the alpha/vignette surfaces have no X/Y offsets.
For 8bit Alpha input, when converted to 16bit output, the 8 bit alpha value is replicated to both the upper and lower 8 bits to form the 16 bit alpha value.
Skin Score Output Surface uses the same tiling format as the Output surface.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Media Command Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VEBOX Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h VEBOX Format: OpCode
	20:16	**SubOpcode B** Default Value: 0h VEBOX Format: OpCode
	15:12	**Reserved** Access: RO Format: MBZ
	11:0	**DWord Length** Format: =n

Value	Name	Description
7h	DWORD_COUNT_n **[Default]**	(Excludes DWords 0, 1)

VEBOX_SURFACE_STATE

1 31:1 **Reserved**

Access:	RO
Format:	MBZ

0 **Surface Identification**

Specifies which set of surfaces this command refers to:

Value	Name
1	Output surface (all except the Denoised Current output surface)
0	Input surface and Denoised Current Output Surface

2 31:18 **Height**

Format:	U14

This field specifies the height of the surface in units of pixels. For PLANAR surface formats, this field indicates the height of the Y (luma) plane.

Value	Name	Description	Exists If
[15, 16383]		representing heights [16,16384]	
[15, 8191]			//Scalar Enabled - For Input surface only
[63, 2047]			//Scalar + SFC Enabled - For Input surface only

Programming Notes
Height (field value + 1) must be a multiple of 2 for PLANAR_420 surfaces. **Height** (field value +1) must be a multiple of 2 when the deinterlace function is enabled (field mode) or when the denoise function is enabled with **Progressive DN** = 0. It must be a multiple of 4 when interleaved deinterlace/denoise and PLANAR_420 are both being used. **VEBOX** supports a minimum height of 16.
Height (field value + 1) must be a multiple of 2 for Bayer surfaces.

17:4 **Width**

Format:	U14

This field specifies the width of the surface in units of pixels. For PLANAR surface formats, this field indicates the width of the Y (luma) plane.

Value	Name	Description	Exists If
[63,16383]		representing widths [64,16384]	
[63,8191]			//Scalar Enabled - For Input surface only
[63,2047]			//Scalar and SFC Enabled - For Input Surface only

VEBOX_SURFACE_STATE

Programming Notes
The Width specified by this field multiplied by the pixel size in bytes must be less than or equal to the surface pitch (specified in bytes via the **Surface Pitch** field). **Width** (field value + 1) must be a multiple of 2 for PLANAR_420, PLANAR_422, and all YCRCB_* surfaces, and must be a multiple of 4 for PLANAR_411 surfaces. **VEBOX** supports a minimum width of 64

3:0 **Reserved**

Access:	RO
Format:	MBZ

3 31:27 **Surface Format**

Format:	U5

Specifies the format of the surface. All of the Y and G channels will use table 0 and all of the Cr/Cb/R/B channels will use table 1.

Value	Name	Description
0	YCRCB_NORMAL	
1	YCRCB_SWAPUVY	
2	YCRCB_SWAPUV	
3	YCRCB_SWAPY	
4	PLANAR_420_8	NV12 with Interleave Chroma set
5	PACKED_444A_8	
6	PACKED_422_16	
7	R10G10B10A2_UNORM / R10G10B10A2_UNORM_SRGB	
8	R8G8B8A8_UNORM / R8G8B8A8_UNORM_SRGB	
9	PACKED_444_16	
10	PLANAR_422_16	
11	Y8_UNORM	
12	PLANAR_420_16	
13	R16G16B16A16	
14	Bayer pattern	
15	Y16_UNORM	

26:25 **Bayer Pattern Offset**

Specifies the starting pixel offset for the Bayer pattern used for Capture Pipe.

Value	Name
00b	Pixel at X=0, Y=0 is Blue
01b	Pixel at X=0, Y=0 is Red
10b	Pixel at X=0, Y=0 is Green, Pixel at X=1, Y=0 is Red
11b	Pixel at X=0, Y=0 is Green, Pixel at X=1, Y=0 is Blue

VEBOX_SURFACE_STATE

24 **Bayer Pattern Format**

Specifies the format of the Bayer Pattern:

Value	Name
0b	8-bit input at a 8-bit stride
1b	16-bit input at a 16-bit stride

23:22 **Bayer Input Alignment**

Format:	U2

Value	Name
00b	MSB aligned data **[Default]**
01b	10bit LSB aligned data
10b	12bit LSB aligned data
11b	14bit LSB aligned data

Programming Notes
Valid only Bayer Pattern Format is 16bit input

21 **Reserved**

Access:	RO
Format:	MBZ

20 **Interleave Chroma**

Format:	Enable

This field indicates that the chroma fields are interleaved in a single plane rather than stored as two separate planes. This field is only used for PLANAR surface formats.

19:3 **Surface Pitch**

Format:	U17

This field specifies the surface pitch in (#Bytes - 1):

Value	Name	Description
[63, 131071]	For other linear surfaces	[64B, 128KB]
[511, 131071]	For X-tiled surface	[512B, 128KB] = [1tile, 256 tiles]
[127, 131071]	For Y-tiled surfaces	[128B,128KB] = [1 tile, 1024 tiles]

Programming Notes
For tiled surfaces, the pitch must be a multiple of the tile width. For linear surfaces, the pitch must be a multiple of 64.If Half Pitch for Chroma is set, this field must be a multiple of two tile widths for tiled surfaces, or a multiple of 2 bytes for linear surfaces.

2 **Half Pitch for Chroma**

Format:	Enable

This field indicates that the chroma plane(s) will use a pitch equal to half the value specified in the **Surface Pitch** field. This field is only used for PLANAR surface formats.

<table>
<tr><th colspan="3">VEBOX_SURFACE_STATE</th></tr>
<tr><td></td><td></td><td><table><tr><th>Programming Notes</th></tr><tr><td>Must be programmed to Zero always as this field is not used</td></tr></table></td></tr>
<tr><td></td><td>1:0</td><td><b>Tile Mode</b>
Indicates the Tile Mode for the Surface.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Linear</td></tr><tr><td>1</td><td>TileS(64K)</td></tr><tr><td>2</td><td>X Major</td></tr><tr><td>3</td><td>Tile F</td></tr></table></td></tr>
<tr><td>4</td><td>31:29</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>28:16</td><td><b>X Offset for U</b><table><tr><td>Format:</td><td>U13</td></tr></table>This field must be zero for the VEBOX surface formats</td></tr>
<tr><td></td><td>15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:0</td><td><b>Y Offset for U</b><table><tr><td>Format:</td><td>U15</td></tr></table>This field specifies the vertical offset in rows from the start (origin) or the Luma(Y) plane to the start (origin) of the U(Cb) plane or the interleaved UV plane if <b>Interleave Chroma</b> is enabled.
This field is only used for PLANAR surface formats.<table><tr><th>Programming Notes</th></tr><tr><td>This field must indicate an even number (bit 0 = 0). This field must be evenly divisible by 4 for Tile-Y surfaces (so the offset points to the start of a cache line) For Planar formats, if the surface is in YS or YF tile modes, the Y Offset for U should be an integral multiple of the Tile height of the Luma plane</td></tr></table></td></tr>
<tr><td>5</td><td>31:29</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>28:16</td><td><b>X Offset for V</b><table><tr><td>Format:</td><td>U13</td></tr></table>This field must be zero for the VEBOX surface formats.</td></tr>
<tr><td></td><td>15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

VEBOX_SURFACE_STATE

	14:0	**Y Offset for V** Format: U15 This field specifies the vertical offset in rows from the start (origin) of the Luma(Y) plane to the start (origin) of the V(Cr) plane. This field is only used for PLANAR surface formats with **Interleave Chroma** disabled. **Programming Notes** This field must indicate an even number (bit 0 = 0).This field must be evenly divisible by 4 for Tile-Y surfaces (so the offset points to the start of a cache line). For Planar formats, if the surface is in YS or YF tile modes, the Y Offset for V should be an integral multiple of the Tile height of the Luma plane
6	31	**Reserved** Access: RO Format: MBZ
	30:16	**X Offset for Frame** Format: U15 This is an offset in X from the Surface Base Address in pixels for all planes using this surface. For U/V planes this is added to the X Offset for U/V. After converting to bytes this must be an integer multiple of cache lines in the tiling mode. This specifies the edge of the frame, so adjacent pixels needed for Denoise/Deinterlace/Demosaic are replicated or mirrored. **Programming Notes** If **Y Offset for Frame** >0 the X Offset must be 0. If memory compression is enabled then this must be an even number of cache lines.
	15	**Reserved** Access: RO Format: MBZ
	14:0	**Y Offset for Frame** Format: U15 This is an offset in Y from the Surface Base Address in pixels for all planes using this surface. For U/V planes this is added to the Y Offset for U/V. After converting to bytes this must be an integer multiple of cache lines in the tiling mode. This specifies the edge of the frame, so adjacent pixels needed for Denoise/Deinterlace/Demosaic are replicated or mirrored. **Programming Notes** If **X Offset for Frame** >0 the Y Offset must be 0. For Planar formats, if the surface is in YS or YF tile modes, the Y Offset for Frame should be an integral multiple of the Tile height.
7	31:27	**Compression Format** Format: **Media Compression Format** Format: **Render Compression Format** Specifies the 5 bit compression format.

VEBOX_SURFACE_STATE

26:17 **Reserved**

Access:	RO
Format:	MBZ

16:0 **Derived Surface Pitch**

Format:	U17

This field specifies the surface pitch in (#Bytes - 1) for the derived surfaces: STMM/Denoise statistic surface is described when the **Surface Identification** bit is 0 (Input Surface). The (Current Denoise Output)/(Previous Denoise Input) surfaces are described when the bit is 1 (Output Surface).

Value	Name	Description	Exists If
[63, 131071]		[64B, 128KB]	[Tiled Surface] == 0
[511, 131071]		[512B, 128KB] = [1tile, 256 tiles]	([Tiled Surface] == 1) AND ([Tile Walk] == 0)
[127, 131071]		[128B,128KB] = [1 tile, 1024 tiles]	([Tiled Surface] == 1) AND ([Tile Walk] == 1)

Programming Notes
In DN Only mode, the pitch for the (Current Denoise Output)/(Previous Denoise Input) and the Surface Pitch must be programed the same.
The pitch must be a multiple of the tile width.

8

31:17 **Reserved**

Access:	RO
Format:	MBZ

16:0 **Surface Pitch for Skin Score Output Surfaces**

Format:	U17

This field specifies the surface pitch in (#Bytes - 1) for the Skin Score Output surface if enabled; This is present only in the output surface format and reserved for Input surface format. The height and width are the same as in the Output surface mentioned above.

Value	Name	Description	Exists If
[63, 131071]		[64B, 128KB]	[Tiled Surface] == 0
[511, 131071]		[512B, 128KB] = [1tile, 256 tiles]	([Tiled Surface] == 1) AND ([Tile Walk] == 0)
[127, 131071]		[128B,128KB] = [1 tile, 1024 tiles]	([Tiled Surface] == 1) AND ([Tile Walk] == 1)

Programming Notes
The pitch must be a multiple of the tile width.

VEBOX_TILING_CONVERT

<table>
<tr><th colspan="3">VEBOX_TILING_CONVERT</th></tr>
<tr><td colspan="3">Source: VideoEnhancementCS
Length Bias: 2</td></tr>
<tr><td colspan="3">This command takes the input surface and writes directly to the output surface at high speed. The surface format and width/height of the input and output must be the same, only the tiling mode and pitch can change.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Command Type</b><table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:27</td><td><b>Pipeline</b><table><tr><td>Default Value:</td><td>2h Media</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>26:24</td><td><b>Command OpCode</b><table><tr><td>Default Value:</td><td>4h VEBOX</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>23:21</td><td><b>SubOpcode A</b><table><tr><td>Default Value:</td><td>0h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>20:16</td><td><b>SubOpcode B</b><table><tr><td>Default Value:</td><td>1h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:0</td><td><b>DWord Length</b><table><tr><td>Format:</td><td>=n</td></tr></table><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>3h</td><td></td><td>(Excludes DWords 0, 1)</td></tr></table></td></tr>
<tr><td>1..2</td><td>63:12</td><td><b>Input Address</b><table><tr><td>Format:</td><td>VIRTUAL_ADDR[63:12]</td></tr></table>Specifies bits 47:12 of the 4Kbyte-aligned frame buffer address for reading the current frame.</td></tr>
<tr><td></td><td>11</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

VEBOX_TILING_CONVERT

	10:0	**Input Surface Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS
3..4 **Programming Notes:** Output address must be different from input address	63:12	**Output Address** Format: VIRTUAL_ADDR[63:12] Specifies bits 47:12 of the 4Kbyte-aligned frame buffer address for writing the current frame.
	11	**Reserved** Access: RO Format: MBZ
	10:0	**Output Surface Control Bits** Format: VEB_DI_IECP_COMMAND_SURFACE_CONTROL_BITS

Wait for Event

MSD_WAIT_FOR_EVENT - Wait for Event

Source:	EuSubFunctionGateway
Length Bias:	1

Send a writeback if Event ID occured after MonitorEvent.

DWord	Bit	Description
0	31:29	**Reserved** Access: RO Format: MBZ
	28:25	**Message Length** Format: U4 Specifies the number of GRF registers sent as the message payload. Value: 1 — Name: One **[Default]** — Description: See MDP_TIMEOUT Timeout Data Payload definition.
	24:20	**Response Length** Format: U5 Specifies the number of GRF registers expected as the message response payload. Value: 0 — Name: Zero **[Default]** — Description: Event completion notification is signaled with ARF N0.0 (bit 0).
	19:3	**Reserved** Access: RO Format: MBZ
	2:0	**Wait for Event Subfunction** Default Value: 0x6 Format: OpCode

While

while - While

Source:	EuIsa
Length Bias:	4
Predication:	true
Conditional Modifier:	false
Saturation:	false
Source Modifier:	false

The while instruction marks the end of a do-while block. The instruction first evaluates the loop termination condition for each channel based on the current channel enables and the predication flags specified in the instruction. If any channel has not terminated, a branch is taken to a destination address specified in the instruction, and the loop continues for those channels. Otherwise, execution continues to the next instruction. Id point to the first instruction with the do label of the do-while block of code. It should be a negative number for the backward referencing. If SPF is ON, none of the PcIP are updated.

Format:

```
[(pred)] while (exec_size) JIP
```

Restriction

The execution size must be the same for the while instruction and any break and cont instructions of the same code block.

Syntax

```
[(pred)] while (exec_size) imm32
```

Pseudocode

```
Evaluate(WrEn);
 for ( n = 0; n < 32; n++ ) {
     if (WrEn.chan[n] ) {
         PcIP[n] = IP + JIP;
     } else {
         PcIP[n] = IP + 1;
     }
 }
 if ( | PMask == 1 ) {  // any enabled channel true
     Jump(IP + JIP);
 }
```

DWord	Bit	Description
0..3	127:96	**Reserved** Exists If: ([Src0.IsImm]==false) Format: MBZ
	127:96	**JIP** Exists If: ([Src0.IsImm]==true) Format: S31

while - While

<table>
<tr><td></td><td></td><td>The byte-aligned jump distance if a jump is taken for the channel</td></tr>
<tr><td></td><td>95:80</td><td><b>Reserved</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>95:64</td><td><b>Reserved</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==true)</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>79:66</td><td><b>Src0.Operand</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>65:64</td><td><b>Reserved</b><table><tr><td>Exists If:</td><td>([Src0.IsImm]==false)</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>63:50</td><td><b>Dst.Operand</b><table><tr><td>Format:</td><td><b>DirectOperand</b></td></tr></table></td></tr>
<tr><td></td><td>49:47</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>46</td><td><b>Src0.IsImm</b>
This field indicate that Source 0 operand is carrying an immediate value.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>false</td></tr><tr><td>1</td><td>true</td></tr></table></td></tr>
<tr><td></td><td>45:34</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>33</td><td><b>BranchCtrl</b>
This field is used by <b><i>goto</i></b>, <b>if</b>, and <b><i>else</i></b> instructions to control branching. See the <b>goto</b> instruction description for more information about BranchCtrl.</td></tr>
<tr><td></td><td>32</td><td><b>AtomicCtrl</b><table><tr><td>Format:</td><td><b>AtomicCtrl</b></td></tr></table></td></tr>
<tr><td></td><td>31</td><td><b>MaskCtrl</b>
Mask Control (formerly Write Enable Control). This field determines if the per channel write enables are used to generate the final write enable. This field should be normally "0".<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Normal <b>[Default]</b></td><td>Normal. Per channel write enable used for final write enable generation.</td></tr></table></td></tr>
</table>

while - While

<table>
<tr><td></td><td></td><td><table><tr><td>1</td><td>NoMask</td><td>NoMask. Skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b></td></tr>
<tr><td></td><td>29</td><td><b>CmptCtrl</b><table><tr><td>Format:</td><td>MBZ</td></tr></table>Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format for more information.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>NoCompaction <b>[Default]</b></td><td>No compaction. 128-bit native instruction supporting all instruction options.</td></tr><tr><td>1</td><td>Compacted</td><td>Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>PredInv</b>
This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 - there is no predication. PMask is the final predication mask produced by the effects of both fields<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td>Positive <b>[Default]</b></td><td>Positive polarity of predication. Use the predication mask produced by PredCtrl.</td></tr><tr><td>1</td><td>Negative</td><td>Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.</td></tr></table></td></tr>
<tr><td></td><td>27:24</td><td><b>PredCtrl</b><table><tr><td>Format:</td><td><b>PredCtrl</b></td></tr></table>This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register.</td></tr>
<tr><td></td><td>23</td><td><b>FlagRegNum[0]</b>
This field specifies bit[0] of the register number for a flag register operand.</td></tr>
<tr><td></td><td>22</td><td><b>FlagSubRegNum</b>
This field specifies the sub-register number for a flag register operand. There are two sub-registers in the flag register. Each sub-register contains 16 flag bits. The selected flag sub-register is the source for predication if predication is enabled for the instruction. It is the destination to store conditional flag bits if conditional modifier is enabled for the instruction. The same flag sub-register can be both the predication source and conditional destination, if both predication and conditional modifier are enabled.</td></tr>
</table>

while - While		
	21:19	**ChanOff** Format: | **ChanOff** This field provides offset information for ARF selection. The can be thought of as a starting channel offset for the execution mask and other ARF registers implicitly accessed.
	18:16	**ExecSize** Format: | **ExecSize** This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	15:0	**Header** Format: | **Header**

Word Atomic Counter with Return Data Operation MSD

MSD1R_WAC - Word Atomic Counter with Return Data Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Ch Format: Opcode Atomic Half Counter Operation message

MSD1R_WAC - Word Atomic Counter with Return Data Operation MSD

	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Access: RO Format: MBZ
	11:8	**Atomic Integer Operation** Format: MDC_AOP Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Word Atomic Counter Write Only Operation MSD

MSD1W_WAC - Word Atomic Counter Write Only Operation MSD

Source: EuSubFunctionDataPort1

Length Bias: 1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header).Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHR Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Ch Format: Opcode Atomic Half Counter Operation message

MSD1W_WAC - Word Atomic Counter Write Only Operation MSD

	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Access: RO Format: MBZ
	11:8	**Atomic Integer Operation** Format: MDC_AOP Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Word Typed Atomic Integer with Return Data Operation MSD

MSD1R_WTAI - Word Typed Atomic Integer with Return Data Operation MSD

Source: EuSubFunctionDataPort1

Length Bias: 1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Format: Opcode Typed Atomic Half Integer Operation message

MSD1R_WTAI - Word Typed Atomic Integer with Return Data Operation MSD

	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved** Access: RO Format: MBZ
	11:8	**Atomic Integer Operation** Format: MDC_AOP Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Word Typed Atomic Integer Write Only Operation MSD

MSD1W_WTAI - Word Typed Atomic Integer Write Only Operation MSD

Source: EuSubFunctionDataPort1

Length Bias: 1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Format: Opcode Typed Atomic Half Integer Operation message

MSD1W_WTAI - Word Typed Atomic Integer Write Only Operation MSD

	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved** Access: RO Format: MBZ
	11:8	**Atomic Integer Operation** Format: MDC_AOP Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS Specifies the Binding Table Index for the message

Word Untyped Atomic Float with Return Data Operation MSD

MSD1R_WAF - Word Untyped Atomic Float with Return Data Operation MSD		
Source:	EuSubFunctionDataPort1	
Length Bias:	1	
DWord	**Bit**	**Description**
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 1Ch Format: Opcode Untyped Atomic Half Float Operation message

MSD1R_WAF - Word Untyped Atomic Float with Return Data Operation MSD

	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11	**Reserved** Access: RO Format: MBZ
	10:8	**Atomic Float Operation** Format: MDC_FOP Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Word Untyped Atomic Float Write Only Operation MSD

MSD1W_WAF - Word Untyped Atomic Float Write Only Operation MSD

Source: EuSubFunctionDataPort1

Length Bias: 1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 1Ch Format: Opcode Untyped Atomic Half Float Operation message

MSD1W_WAF - Word Untyped Atomic Float Write Only Operation MSD

	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11	**Reserved** Access: RO Format: MBZ
	10:8	**Atomic Float Operation** Format: MDC_FOP Specifies the atomic float operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Word Untyped Atomic Integer with Return Data Operation MSD

MSD1R_WAI - Word Untyped Atomic Integer with Return Data Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Format: Opcode Untyped Atomic Half Integer Operation message

MSD1R_WAI - Word Untyped Atomic Integer with Return Data Operation MSD

	13	**Return Data Control** Default Value: 1h Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Format: MDC_SM2R Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: MDC_AOP Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: MDC_BTS_SLM_A32 Specifies the Binding Table Index for the message

Word Untyped Atomic Integer Write Only Operation MSD

MSD1W_WAI - Word Untyped Atomic Integer Write Only Operation MSD

Source:	EuSubFunctionDataPort1
Length Bias:	1

DWord	Bit	Description
0	31	**Reserved** Access: RO Format: MBZ
	30	**Packed Data Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD data item in the source and writeback data payload is stored in GRF as a 32-bit value (8 per GRF), and smaller data items are zero padded to 32 bits. When set, each SIMD data item in the source and writeback data payload is stored in GRF as a 16-bit value (16 per GRF). **Restriction** Only 32-bit data packing is supported at this time.
	29	**Packed Address Payload** Default Value: 0 32 bit Format: Enable When clear, each SIMD address in the address payload is stored in GRF as a 32-bit value (8 per GRF). When set, each SIMD address in the address payload is stored in GRF as a 16-bit value (16 per GRF). Addresses in message header payloads are always stored as 32-bit values.
	28:25	**Message Length** Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Format: MDC_MHP If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Format: Opcode Untyped Atomic Half Integer Operation message

MSD1W_WAI - Word Untyped Atomic Integer Write Only Operation MSD

	13	**Return Data Control** Default Value: 0h Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Format: **MDC_SM2R** Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Format: **MDC_AOP** Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Format: **MDC_BTS_SLM_A32** Specifies the Binding Table Index for the message

XY_BLOCK_COPY_BLT

XY_BLOCK_COPY_BLT

Source:	BlitterCS
Length Bias:	2

Description

XY_BLOCK_COPY_BLT instruction performs a color source copy where the only operands involved are a color source and destination of the same bit width. The source and destination surfaces CAN overlap, the hardware handles this internally. Legacy blit commands (2D BLT instructions other than XY_BLOCK_COPY_BLT, XY_FAST_COPY_BLT, XY_FAST_COLOR_BLT) and this new copy command can be interspersed. No implied flush required between the two provided there is no producer consumer relationship between the two. The starting pixel of the blit operation for both source and destination should be on a pixel boundary. This command now supports copy of compressed surface.
In case of producer consumer relationship between a legacy blitter command and anew copy command a flush must be inserted between the two by software.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Client</b>
<table>
<tr><td>Default Value:</td><td>02h 2D Processor</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table>
<tr><td>Format:</td><td>Opcode</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>41h</td><td>INSTRUCTION_TARGET_XY_BLOCK_COPY_BLT <b>[Default]</b></td></tr>
</table></td></tr>
<tr><td></td><td>21:19</td><td><b>Color Depth</b>
This field actually programs bits per pixel value for each pixel of the surface. Reprogramming of these bits require explicit flushing of Copy Engine.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>000b</td><td>8 bit color <b>[Default]</b></td></tr>
<tr><td>001b</td><td>16 bit color</td></tr>
<tr><td>010b</td><td>32 bit color</td></tr>
<tr><td>011b</td><td>64 bit color</td></tr>
<tr><td>100b</td><td>96 bit color (only linear case is supported)</td></tr>
<tr><td>101b</td><td>128 bit color</td></tr>
<tr><td>110b</td><td>RESERVED</td></tr>
<tr><td>111b</td><td>RESERVED</td></tr>
</table></td></tr>
</table>

XY_BLOCK_COPY_BLT

		Programming Notes Color depth programming for 96 bit color is invalid unless both the source and destination surfaces are linear.
	18:14	**Reserved** Access: RO Format: MBZ
	13:12	**Special Mode of Operation** This field indicates the mode of operation for the command.

Value	Name	Description	Programming Notes
0h	NONE **[Default]**	No special mode. It will act as regular copy command.	Destination AUX may or may not be enabled depending on whether the surface is compressible or not.
1h	FULL_RESOLVE	In-place resolve to get rid of 128B blocks from clear or compression state.	If Resolve Mode is programmed as FULL_RESOLVE, AUX for destination surface must be enabled. When special mode of operation is set as FULL_RESOLVE destination surface is fully decompressed irrespective of the compression enable setting. In order to resolve a given rectangular surface both source and destination rectangle must be programmed with same overlapping values (source and destination to 100% overlap) and special operation mode must be programmed to FULL_RESOLVE.

XY_BLOCK_COPY_BLT

2h	PARTIAL_RESOLVE	Partial resolve is for resolving the surface for clear values. If the surface is compressed it keeps it compressed, no implied clear values.	
3h	Reserved	Reserved for future use.	

11:9

Number of Multisamples

This filed indicates number of multi-samples on the surface.

Value	Name
000b	MULTISAMPLECOUNT_1 **[Default]**
001b	MULTISAMPLECOUNT_2
010b	MULTISAMPLECOUNT_4
011b	MULTISAMPLECOUNT_8
100b	MULTISAMPLECOUNT_16
101b	RESERVED
110b	RESERVED
111b	RESERVED

Programming Notes
Currently the number of samples supported by Copy Engine is only MULTISAMPLECOUNT_1 . Rest of the values are reserved for future projects.

8

Reserved

Access:	RO
Format:	MBZ

7:0

DWord Length

Format:	=n

Description
n = 20 This field indicates length of the instruction in DWORD.

Value	Name
20	Excludes DWORD 0,1 **[Default]**

<table>
<tr><th colspan="3">XY_BLOCK_COPY_BLT</th></tr>
<tr><td>1</td><td>31:30</td><td><b>Destination Tiling</b>
These bits indicate destination tiling method.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Programming Notes</th></tr>
<tr><td>00b</td><td>LINEAR <b>[Default]</b></td><td>Linear mode (no tiling)</td><td></td></tr>
<tr><td>01b</td><td>XMAJOR</td><td>X major Tiling</td><td></td></tr>
<tr><td>10b</td><td>Tile4</td><td>Tile4 4KB tiling</td><td></td></tr>
<tr><td>11b</td><td>Tile64</td><td>Tile64 64KB tiling</td><td>Tile64 is not supported if surface type is 3D</td></tr>
</table>
</td></tr>
<tr><td></td><td>29</td><td><b>Destination Compression Enable</b>
Selects the type of write operation (compressed/uncompressed) to the destination surface.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Programming Notes</th></tr>
<tr><td>0b</td><td>Compression Disable <b>[Default]</b></td><td>Enables uncompressed write operation to destination surface.</td><td>If Compression is disabled AUX may or may not be enabled.
If AUX is enabled, which indicates that destination surface is a compressible surface, writes to the destination surface is controlled by the Resolve Mode. If special mode of operation is PARTIAL_RESOLVE, compression can't be disabled.</td></tr>
<tr><td>1b</td><td>Compression Enable</td><td>Enables compressed write operation to destination surface provided special mode of operation is not FULL_RESOLVE.</td><td>Compression Enable require AUX to be enabled.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Destination compression can be enabled irrespective of the value programmed in the destination target memory field (LOCAL_MEM or SYSTEM_MEM) as the value programmed in that field is considered as only performance hint.</td></tr>
</table>
</td></tr>
</table>

XY_BLOCK_COPY_BLT

28 **Destination Control Surface Type**

Value	Name	Description	Programming Notes
0b	3D Control Surface **[Default]**	Control Surface type is 3D.	
1b	Media Control Surface	Control Surface type is media.	When destination compression is enabled associated control surface type cannot be media.

27:21 **Destination MOCS value**

MOCS (Memory Object State Control) for destination operand.

Programming Notes
Destination MOCS value, which is used to program MOCS index for writing to memory, should select a MOCS register having "L3 Cacheability Control" programmed as uncacheable(UC) and "Global GO" parameter set as GO Memory (pushes GO point to memory). The MOCS Register may have L3 Lookup programmed as UCL3LKDIS for better efficiency.

20:18 **Destination Auxiliary surface mode**

Format:	U3

Specifies type of the AUX surface associated with the primary surface (destination).

Value	Name	Description
000b	AUX_NONE **[Default]**	No Auxiliary surface used
001b	Reserved	
010b	Reserved	
011b	Reserved	
100b	Reserved	
101b	AUX_CCS_E	Auxiliary surface is a CCS with lossless compression enabled when number of multisamples is 1. When number of multisamples > 1, programming this value means MSAA compression enabled.
110b	Reserved	
111b	Reserved	

Programming Notes
Only AUX_NONE and AUX_CCS_E values are valid now, rest of the values are reserved for future projects.

XY_BLOCK_COPY_BLT

<table>
<tr><td></td><td>17:0</td><td><b>Destination Pitch</b><table><tr><td>Format:</td><td>U18-1</td></tr></table>For <b>Linear Surfaces</b>, the pitch must be multiple of pixel width in bytes. For Tiled surfaces, the pitch has to be a <b>multiple of the Tile Width</b> (X direction width of the Tile) expressed in dwords (4 bytes).</td></tr>
<tr><td>2</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16-bit signed number. The destination start line (inclusive) for Block Copy blit.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16-bit signed number. The destination start pixel (inclusive) for Block Copy blit.</td></tr>
<tr><td>3</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
16-bit signed number. The destination end line (exclusive) for Block Copy blit.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X2 Coordinate (Right)</b>
16-bit signed number. The destination end pixel (exclusive) for Block Copy blit.</td></tr>
<tr><td>4..5</td><td>63:0</td><td><b>Destination Base Address</b><table><tr><td>Format:</td><td>GraphicsAddress[63:0]</td></tr></table><table><tr><th>Description</th></tr><tr><td>This bitfield contains the base address of the destination surface.
When Tiling is enabled this address is cacheline (64Byte) aligned.
When Destination Tiling is disabled, this address is byte aligned.</td></tr></table></td></tr>
<tr><td>6</td><td>31</td><td><b>Destination Target Memory</b>
Target memory for destination. It can be local memory or system memory.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0b</td><td>LOCAL_MEM <b>[Default]</b></td><td>Target memory is local memory.</td></tr><tr><td>1b</td><td>SYSTEM_MEM</td><td>Target memory is system memory.</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>This field is used just as performance hint for destination target memory. If this bit is set as SYSTEM_MEM in the command, the writes will have target memory field set accordingly and those writes are given less priority in the system. When the number of such outstanding writes in the system crosses a certain threshold, copy engine is throttled. Please check registers 0x22200[15:2] (BCS SW Control) and 0x220A0[15:10] (Mode Register for GAB) for further details.</td></tr></table></td></tr>
</table>

XY_BLOCK_COPY_BLT

	30	**Reserved** Access: RO Format: MBZ
	29:16	**Destination Y offset** **Description** Format is U14. This field specifies the vertical offset in lines from the Surface Base Address to the start (origin) of the surface. **Programming Notes** For Linear surface Y offset must be 0. For Tiled surface Y offset must be greater than or equal to 0 and less than 16K in lines.
	15:14	**Reserved** Access: RO Format: MBZ
	13:0	**Destination X offset** **Description** Format is U14. This field specifies the horizontal offset in pixels from the surface base address to the start (origin) of the surface. **Programming Notes** For Linear surface X offset must be 0. For Tiled surface X offset must be greater than or equal to 0 and less than 16K in pixels.
7	31:16	**Source Y1 Coordinate (Top)** Format is S16. The source start line (inclusive) for the Block Copy blit.
	15:0	**Source X1 Coordinate (Left)** Format is S16. Source start pixel (inclusive) for Block Copy blit.
8	31:30	**Source Tiling** **Description** These bits indicate source tiling method.

XY_BLOCK_COPY_BLT

Value	Name	Description	Programming Notes
00b	LINEAR **[Default]**	Linear Tiling (tiking disabled)	
01b	XMAJOR	X major tiling	
10b	Tile4	Tile4 4KB tiling	
11b	Tile64	Tile64 64KB tiling	Tile64 is not supported if surface type is 3D

29 **Source Compression Enable**

Enable reading of compressed data from source surface.

Value	Name	Programming Notes
0b	Compression disable **[Default]**	AUX may or may not be enabled. If AUX is enabled and Compression is disabled in that case user must ensure that the source surface is already fully resolved in order to perform uncompressed read of this compressible surface correctly.
1b	Compression enable	Compression Enable require AUX to enabled.

28 **Source Control Surface Type**

Value	Name	Description
0b	3D Control Surface **[Default]**	Control Surface type is 3D.
1b	Media Control Surface	Control Surface type is media

27:21 **Source MOCS**

Description
MOCS (Memory Object State Control) value for source operand.

Programming Notes
Source MOCS value, which is used to program MOCS index for reading from memory, should select a MOCS register having "L3 Cacheability Control" programmed as uncacheable(UC). The MOCS Register may have L3 Lookup programmed as UCL3LKDIS for better efficiency.

20:18 **Source Auxiliary surface mode**

Format:	U3

Specifies type of the AUX surface associated with the primary surface (source).

XY_BLOCK_COPY_BLT

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>000b</td><td>AUX_NONE</td><td>No Auxiliary surface used</td></tr>
<tr><td>001b</td><td>Reserved</td><td></td></tr>
<tr><td>010b</td><td>Reserved</td><td></td></tr>
<tr><td>011b</td><td>Reserved</td><td></td></tr>
<tr><td>100b</td><td>Reserved</td><td></td></tr>
<tr><td>101b</td><td>AUX_CCS_E</td><td>Auxiliary surface is a CCS with lossless compression enabled when number of multisamples is 1. When number of multisamples > 1, programming this value means MSAA compression enabled.</td></tr>
<tr><td>110b</td><td>Reserved</td><td></td></tr>
<tr><td>111b</td><td>Reserved</td><td></td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Only AUX_NONE and AUX_CCS_E values are valid now, rest of the values are reserved for future projects.</td></tr>
</table>
</td></tr>
<tr><td></td><td>17:0</td><td>
<b>Source Pitch</b>
<table>
<tr><td>Format:</td><td>U18-1</td></tr>
</table>
For <b>Linear surfaces</b>, the pitch must be multiple of pixel width in bytes. For tiled surfaces, the pitch has to be multiple of the Tile width (X direction width of the tile) expressed in dwords (4 byte).
</td></tr>
<tr><td>9..10</td><td>63:0</td><td>
<b>Source Base Address</b>
<table>
<tr><td>Format:</td><td>GraphicsAddress[63:0]</td></tr>
</table>
This bitfield contains the base address of the source surface. When Tiling is enabled this address is cacheline (64Byte) aligned. When Source Tiling is disabled, this address is byte aligned.
</td></tr>
<tr><td>11</td><td>31</td><td>
<b>Source Target Memory</b>
Target memory for source. It can the local memory or system memory.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>LOCAL_MEM <b>[Default]</b></td><td>Target memory is local memory.</td></tr>
<tr><td>1b</td><td>SYSTEM_MEM</td><td>Target memory is system memory.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This field is used just as performance hint for source target memory. If this bit is set as SYSTEM_MEM in the command, the read requests will have target memory field set accordingly and those reads are given less priority in the system. When the number of such outstanding reads in the system crosses a certain threshold, copy engine is throttled. Please check registers 0x22200[15:2] (BCS SW Control)and 0x220A0[15:10] (Mode Register for GAB) for further details.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th colspan="3">XY_BLOCK_COPY_BLT</th></tr>
<tr><td rowspan="4"></td><td>30</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>29:16</td><td><b>Source Y offset</b>
Format U14.
This field specifies the vertical offset in lines from the Surface Base Address to the start (origin) of the surface.<table><tr><th>Programming Notes</th></tr><tr><td>For Linear surface Y offset must be 0.
For Tiled surface Y offset must be greater than or equal to 0 and less than 16K in lines.</td></tr></table></td></tr>
<tr><td>15:14</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>13:0</td><td><b>Source X offset</b>
Format U14.
This field specifies the horizontal offset in pixels from Surface Base Address to the start (origin) of the surface.<table><tr><th>Programming Notes</th></tr><tr><td>For Linear surface X offset must be 0.
For Tiled surface X offset must be greater than or equal to 0 and less than 16K in pixels.</td></tr></table></td></tr>
<tr><td rowspan="2">12
This field indicates pixel or texel format of the source surface to be copied used by compression/decompression logic.</td><td>31:6</td><td><b>Source Clear Address Low</b><table><tr><td>Format:</td><td>GraphicsAddress[31:6]</td></tr></table>Specifies the lower bits of Graphics Address where clear value is stored in for the source surface. The memory layout of the clear color pointed to by this address is a value stored in the lower-order bytes of the 64-byte cache-line.</td></tr>
<tr><td>5</td><td><b>Source Clear Value Enable</b>
This field indicates whether there is valid clear value available for the source surface. If enabled clear values are stored in the address pointed to by the Source Clear Address field<table><tr><th>Value</th><th>Name</th></tr><tr><td>0b</td><td>Disable <b>[Default]</b></td></tr><tr><td>1b</td><td>Enable</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>Clear value can not be enabled when Color Depth is 96 BPP.</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_BLOCK_COPY_BLT</th></tr>
<tr><td></td><td>4:0</td><td><b>Source Compression Format</b><table><tr><th>Programming Notes</th></tr><tr><td>Compression format definition can be found in Enumeration_RenderCompressionFormat</td></tr><tr><td>For Unified Losless Compression, the compression format definition can be found in Enumeration_RenderCompressionFormat</td></tr></table></td></tr>
<tr><td rowspan="2">13</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>15:0</td><td><b>Source Clear Address High</b><table><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table>Specifies the higher bits of Graphics Address for source surface where clear value is stored in the form of RGBA (R in the LSB and A in the MSB - in that order).</td></tr>
<tr><td rowspan="3">14
This field indicates pixel or texel format of the destination surface used by compression/decompression logic.</td><td>31:6</td><td><b>Destination Clear Address Low</b><table><tr><td>Format:</td><td>GraphicsAddress[31:6]</td></tr></table>Specifies the lower bits of Graphics Address where clear value is stored in for the destination surface. The memory layout of the clear color pointed to by this address is a value stored in the lower-order bytes of the 64-byte cache-line.</td></tr>
<tr><td>5</td><td><b>Destination Clear Value Enable</b>
This field indicates whether there is valid clear value available for the destination surface. If enabled clear values are stored in the address pointed to by the Source Clear Address field.<table><tr><th>Value</th><th>Name</th></tr><tr><td>0b</td><td>Disable <b>[Default]</b></td></tr><tr><td>1b</td><td>Enable</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td>Clear value cannot be enabled when Color Depth is 96 BPP.</td></tr></table></td></tr>
<tr><td>4:0</td><td><b>Destination Compression Format</b><table><tr><th>Programming Notes</th></tr><tr><td>Compression Format definition can be found in Enumeration_RenderCompressionFormat</td></tr><tr><td>For Unified Losless Compression, compression format definition can be found in Enumeration_RenderCompressionFormat</td></tr></table></td></tr>
<tr><td>15</td><td>31:16</td><td><b>Reserved</b><table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_BLOCK_COPY_BLT</th></tr>
<tr><td></td><td>15:0</td><td><b>Destination Clear Address High</b>
<table><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table>
Specifies the higher bits of Graphics Address for destination surface where clear value is stored in the form of RGBA (R in the LSB and A in the MSB - in that order).</td></tr>
<tr><td>16</td><td>31:29</td><td><b>Destination Surface Type</b>
This field defines type of the destination surface.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>SURFTYPE_1D <b>[Default]</b></td><td>Defines a 1-dimensional map or array of maps</td></tr>
<tr><td>1h</td><td>SURFTYPE_2D</td><td>Defines a 2-dimensional map or array of maps</td></tr>
<tr><td>2h</td><td>SURFTYPE_3D</td><td>Defines a 3-dimensional (volumetric) map</td></tr>
<tr><td>3h</td><td>SURFTYPE_CUBE</td><td>Defines a cube map or array of cube maps.</td></tr>
<tr><td>4h</td><td>Reserved</td><td></td></tr>
<tr><td>5h</td><td>Reserved</td><td></td></tr>
<tr><td>6h</td><td>Reserved</td><td></td></tr>
<tr><td>7h</td><td>Reserved</td><td></td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>27:14</td><td><b>Destination Surface Width</b>
<table><tr><td>Format:</td><td>U14-1</td></tr></table>
This field specifies the width of the destination surface, minus 1. If the surface is MIP-mapped, this field specifies the width of the base MIP level. The width is specified in units of pixels or texels.</td></tr>
<tr><td></td><td>13:0</td><td><b>Destination Surface Height</b>
<table><tr><td>Format:</td><td>U14-1</td></tr></table>
This field specifies the height of the destination surface, minus 1. If the surface is MIP-mapped, this field contains the height of the base MIP level.</td></tr>
<tr><td>17</td><td>31:21</td><td><b>Destination Surface Depth</b>
<table><tr><td>Format:</td><td>U11-1</td></tr></table>
This field specifies the total number of levels, minus 1, for a volume texture or the number of array elements, minus 1, allowed to be accessed starting at the Minimum Array Element for arrayed surfaces. If the volume texture is MIP-mapped, this field specifies the depth of the base MIP level.</td></tr>
</table>

<table>
<tr><th colspan="3">XY_BLOCK_COPY_BLT</th></tr>
<tr><td></td><td>20:19</td><td><b>Reserved</b><table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>18:4</td><td><b>Destination Surface Qpitch</b>
The interpretation of this field is dependent on Surface Type as follows:<ul><li>SURFTYPE_1D: distance in <i>pixels</i> between array slices</li><li>SURFTYPE_2D/CUBE: distance in <i>rows</i> between array slices. For Quilted Textures this field specifies the distance in rows between <i>quilt</i> slices. For compressed texture formats, one row contains a complete compression block vertically.</li><li>SURFTYPE_3D: distance in <i>rows</i> between R-slices [<b>Note:</b> these <i>rows</i> are only in the vertical dimension without considering the depth dimension]. For compressed texture formats, one row contains a complete compression block vertically.</li><li>Other surface types: field is ignored</li></ul><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>[1h,7FFFh]</td><td></td><td>in multiples of 4 (low 2 bits missing). The actual qpitch value is 4 times the value programmed.</td></tr></table><table><tr><th>Programming Notes</th></tr><tr><td><b>For Surface Type 1D:</b> This field must be set to an integer multiple of the <b>Surface Horizontal Alignment</b>
<b>For Surface Type 2D, CUBE:</b> This field must be set to an integer multiple of the <b>Surface Vertical Alignment</b>
<b>For Surface Type 3D:</b> <i>Tile Mode != Linear:</i> This field must be set to an integer multiple of the tile height(2^Cv). <i>Tile Mode == Linear:</i> This field must be set to an integer multiple of the Surface Vertical Alignment</td></tr></table></td></tr>
<tr><td></td><td>3:0</td><td><b>Destination LOD</b><table><tr><td>Default Value:</td><td>[0h, Fh]</td></tr></table>LOD of the destination surface to be copied.<table><tr><th>Programming Notes</th></tr><tr><td>Default value of destination LOD is 0. This value must be programmed to 0 for non-MIP mapped destination surfaces.</td></tr></table></td></tr>
<tr><td>18</td><td>31:21</td><td><b>Destination Array Index</b><table><tr><td>Format:</td><td>U11-1</td></tr></table>For arrayed surfaces (2D arrays, 1D arrays or cube arrays) this indicates the array index. For MIP MAPPED and volumetric surface this indicates the slice index of the destination surface to be copied.</td></tr>
<tr><td></td><td>20:19</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_BLOCK_COPY_BLT</th></tr>
<tr><td></td><td>18</td><td><b>Destination Depth/Stencil Resource</b>
This bit field, when set, indicates if the resource is created as Depth/Stencil resource.</td></tr>
<tr><td></td><td>17:12</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11:8</td><td><b>Destination Mip Tail Start LOD</b>
This field indicates which LOD is the first one in the MIP tail if if <b>Tiled Mode</b> is programmed to Tile64. The MIP tail has a different layout than the rest of the surface. Refer to the <i>Memory Data Formats</i> section for more details. <b>For other tiled formats and linear surfaces</b> this field is ignored.</td></tr>
<tr><td></td><td>7:5</td><td><b>Reserved</b><table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>4:3</td><td><b>Destination Vertical Align</b><table><tr><th>Description</th></tr><tr><td>This field specifies the vertical alignment requirement in elements for destination surface.
This field is used for 2D, CUBE, and 3D surface alignment when Tiled Mode is not Tile64. It is ignored for Tile64 surfaces. It is also ignored for 1D surfaces.
See the appropriate Alignment table in the "Surface Layout and Tiling" section under Common Surface Formats for the table of alignment values for Tile64.
Further details can be found in Structure_RENDER_SURFACE_STATE.</td></tr></table></td></tr>
<tr><td></td><td>2</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>1:0</td><td><b>Destination Horizontal Align</b>
This field is used for horizontal alignment of destination surface. Details regarding HALIGN field can be found in surface state description area in Structure_RENDER_SURFACE_STATE.</td></tr>
<tr><td>19</td><td>31:29</td><td><b>Source Surface Type</b>
This field defines type of the source surface.<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>SURFTYPE_1D <b>[Default]</b></td><td>Defines a 1-dimensional map or array of maps</td></tr><tr><td>1h</td><td>SURFTYPE_2D</td><td>Defines a 2-dimensional map or array of maps</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_BLOCK_COPY_BLT</th></tr>
<tr><td></td><td></td><td><table>
<tr><td>2h</td><td>SURFTYPE_3D</td><td>Defines a 1-dimensional (volumetric) map.</td></tr>
<tr><td>3h</td><td>SURFTYPE_CUBE</td><td>Defines a cube map or array of cube maps.</td></tr>
<tr><td>4h</td><td>Reserved</td><td></td></tr>
<tr><td>5h</td><td>Reserved</td><td></td></tr>
<tr><td>6h</td><td>Reserved</td><td></td></tr>
<tr><td>7h</td><td>Reserved</td><td></td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>Reserved</b><table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>27:14</td><td><b>Source Surface Width</b><table>
<tr><td>Format:</td><td>U14-1</td></tr>
</table>This field specifies the width of the surface, minus 1. If the surface is MIP-mapped, this field specifies the width of the base MIP level. The width is specified in units of pixels or texels.</td></tr>
<tr><td></td><td>13:0</td><td><b>Source Surface Height</b><table>
<tr><td>Format:</td><td>U14-1</td></tr>
</table>This field specifies the height of the surface, minus 1. If the surface is MIP-mapped, this field contains the height of the base MIP level.</td></tr>
<tr><td>20</td><td>31:21</td><td><b>Source Surface Depth</b><table>
<tr><td>Format:</td><td>U11-1</td></tr>
</table>This field specifies the total number of levels, minus 1, for a volume texture or the number of array elements, minus 1, allowed to be accessed starting at the Minimum Array Element for arrayed surfaces. If the volume texture is MIP-mapped, this field specifies the depth of the base MIP level.</td></tr>
<tr><td></td><td>20:19</td><td><b>Reserved</b><table>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>18:4</td><td><b>Source Surface Qpitch</b>
The interpretation of this field is dependent on Surface Type as follows:<ul>
<li>SURFTYPE_1D: distance in <i>pixels</i> between array slices</li>
<li>SURFTYPE_2D/CUBE: distance in <i>rows</i> between array slices. For Quilted Textures this field specifies the distance in rows between <i>quilt</i> slices. For compressed texture formats, one row contains a complete compression block vertically.</li>
<li>SURFTYPE_3D: distance in <i>rows</i> between R-slices [<b>Note:</b> these <i>rows</i> are only in the vertical dimension without considering the depth dimension]. For compressed texture formats, one row contains a complete compression block vertically.</li>
<li>Other surface types: field is ignored</li>
</ul></td></tr>
</table>

<table>
<tr><th colspan="3">XY_BLOCK_COPY_BLT</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>[1h,7FFFh]</td><td></td><td>in multiples of 4 (low 2 bits missing). The actual Qpitch value is 4 times the value programmed.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td><b>For Surface Type 1D:</b>This field must be set to an integer multiple of the <b>Surface Horizontal Alignment</b>
<b>For Surface Type 2D, CUBE:</b> This field must be set to an integer multiple of the <b>Surface Vertical Alignment</b>
<b>For Surface Type 3D:</b><i>Tile Mode != Linear:</i> This field must be set to an integer multiple of the tile height(2^Cv).<i>Tile Mode == Linear:</i> This field must be set to an integer multiple of the Surface Vertical Alignment</td></tr>
</table>
</td></tr>
<tr><td></td><td>3:0</td><td><b>Source LOD</b>
<table>
<tr><td>Default Value:</td><td>[0h, Fh]</td></tr>
</table>
LOD of the source surface to be copied.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Default value of source LOD is 0. This value must be programmed to 0 for non-MIP mapped source surfaces.</td></tr>
</table>
</td></tr>
<tr><td>21</td><td>31:21</td><td><b>Source Array Index</b>
<table>
<tr><td>Format:</td><td>U11-1</td></tr>
</table>
For arrayed surfaces (2D arrays, 1D arrays or cube arrays) this indicates the array index. For MIP MAPPED and volumetric surface this indicates the slice index of the source surface to be copied.</td></tr>
<tr><td></td><td>20:19</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>18</td><td><b>Source Depth/Stencil Resource</b>
This bit field, when set, indicates if the resource is created as Depth/Stencil resource.</td></tr>
<tr><td></td><td>17:12</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>11:8</td><td><b>Source Mip Tail Start LOD</b>
This field indicates which LOD is the first one in the MIP tail if <b>Tiled Resource Mode</b> is not TRMODE_NONE. The MIP tail has a different layout than the rest of the surface.</td></tr>
<tr><td></td><td>7:5</td><td><b>Reserved</b>
<table>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td></td><td>4:3</td><td><b>Source Vertical Align</b></td></tr>
</table>

XY_BLOCK_COPY_BLT		
		Description This field is used for 2D, CUBE, and 3D surface alignment when Tiled Mode is not Tile64. It is ignored for Tile64 surfaces. It is also ignored for 1D surfaces. See the appropriate Alignment table in the "Surface Layout and Tiling" section under Common Surface Formats for the table of alignment values for Tile64. Further details about the field can be found in Structure_RENDER_SURFACE_STATE.
	2	**Reserved** Access: RO Format: MBZ
	1:0	**Source Horizontal Align** This field is used for horizontal alignment of source surface. Details regarding HALIGN field can be found in surface state description area in Structure_RENDER_SURFACE_STATE.

XY_COLOR_BLT

<table>
<tr><th colspan="3">XY_COLOR_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">COLOR_BLT is the simplest BLT operation. It performs a color fill to the destination (with a possible ROP). The only operand is the destination operand which is written dependent on the raster operation. The solid pattern color is stored in the pattern background register.
This instruction is optimized to run at the maximum memory write bandwidth.
The typical (and fastest) Raster operation code = F0 which performs a copy of the pattern background register to the destination.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0
BR00</td><td>31:29</td><td><b>Client</b>
<table>
<tr><td>Default Value:</td><td>02h 2D Processor</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table>
<tr><td>Default Value:</td><td>50h</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>21:20</td><td><b>32bpp Byte Mask</b>
This field is only used for 32bpp.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>1xb</td><td>Write Alpha Channel</td></tr>
<tr><td>x1b</td><td>Write RGB Channel</td></tr>
</table></td></tr>
<tr><td></td><td>19:12</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>11</td><td><b>Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>Tile-X or Tile-Y</td></tr>
</table></td></tr>
<tr><td></td><td>10:8</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Default Value:</td><td>05h</td></tr>
<tr><td>Format:</td><td>=n</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_COLOR_BLT</th></tr>
<tr><td rowspan="6">1
BR13</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>30</td><td><b>Clipping Enabled</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Disabled</td></tr>
<tr><td>1b</td><td>Enabled</td></tr>
</table></td></tr>
<tr><td>29:26</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>25:24</td><td><b>Color Depth</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>8 Bit Color</td></tr>
<tr><td>01b</td><td>16 Bit Color(565)</td></tr>
<tr><td>10b</td><td>16 Bit Color(1555)</td></tr>
<tr><td>11b</td><td>32 Bit Color</td></tr>
</table></td></tr>
<tr><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement
For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be up to 128Kbytes (or 32KDwords).</td></tr>
<tr><td rowspan="2">2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
<tr><td rowspan="2">3
BR23</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
16 bit signed number.</td></tr>
<tr><td>15:0</td><td><b>Destination X2 Coordinate (Right)</b>
16 bit signed number.</td></tr>
<tr><td>4..5</td><td>63:0</td><td><b>Destination Base Address</b>
<table>
<tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr>
</table>
This address must be Cache Line (64Byte) aligned for all surface types (linear or tiled).</td></tr>
<tr><td>6
BR16</td><td>31:0</td><td><b>Solid Pattern Color</b>
8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]</td></tr>
</table>

XY_CTRL_SURF_COPY_BLT

<table>
<tr><th colspan="3">XY_CTRL_SURF_COPY_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">XY_CTRL_SURF_COPY_BLT instruction copies control surface associated with a main surface from source to destination. This operation is always associated with main surface copy operation.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Client</b>
<table>
<tr><td>Default Value:</td><td>02h 2D Processor</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>28:22</td><td><b>Instruction Target(opcode)</b>
<table>
<tr><td>Format:</td><td>Opcode</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>48h</td><td><b>[Default]</b></td></tr>
</table></td></tr>
<tr><td></td><td>21</td><td><b>Source Access Type</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>INDIRECT_ACCESS <b>[Default]</b></td><td>Address used to access CCS is the virtual address of the associated main surface.</td></tr>
<tr><td>1</td><td>DIRECT_ACCESS</td><td>Address used is the virtual address of the CCS. Address is used directly.</td></tr>
</table></td></tr>
<tr><td></td><td>20</td><td><b>Destination Access Type</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>INDIRECT_ACCESS <b>[Default]</b></td><td>Address used to access CCS is the virtual address of the associated main surface.</td></tr>
<tr><td>1</td><td>DIRECT_ACCESS</td><td>Address used is the virtual address of the CCS. Address is used directly.</td></tr>
</table></td></tr>
<tr><td></td><td>19:18</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>17:8</td><td><b>Size of Control Surface Copy</b>
<table>
<tr><td>Format:</td><td>U10-1</td></tr>
</table>
This field indicates size of the Control Surface or CCS copy. It is expressed in terms of number of 256B block of CCS, where each 256B block of CCS corresponds to 64KB of main surface.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>[0b,1111111111b]</td><td>COPY_SIZE</td><td>The programmed value is one less than the number of 256B CCS block that is intended to be copied.</td></tr>
</table></td></tr>
</table>

XY_CTRL_SURF_COPY_BLT

	7:0	**DWord Length** Default Value: | 3h Excludes DWORD 0, 1 Format: | =n Indicates the length of the instruction in DWORD.
1	31:12	**Source Start Address Low** Format: | GraphicsAddress[31:12] Specifies the lower bits of Graphics Address for the Start of the main surface whose CCS has to be copied, if Source Access Type is indirect access. If the Source Access Type is direct access this field indicates the CCS address in the native address space (actual virtual address within System Memory). It is 64K aligned if the access type is indirect access and 4K aligned if access type is direct access.
	11:0	**Reserved** Access: | RO Format: | MBZ
2	31:25	**Source MOCS** MOCS (Memory Object State Control) for source operand. **Programming Notes** Source MOCS value, which is used to program MOCS index for reading from memory, should select a MOCS register having "L3 Cacheability Control" programmed as uncacheable(UC). The MOCS Register may have L3 Lookup programmed as UCL3LKDIS for better efficiency.
	24:16	**Reserved** Format: | MBZ
	15:0	**Source Start Address High** Format: | GraphicsAddress[47:32] Specifies the higher bits of Graphics Address for the Start of the main surface whose CCS has to be copied, if the Source Access type is Indirect Access. If the Source Access type is direct access this field indicates the higher address bits of the CCS address in the native address space.
3	31:12	**Destination Start Address Low** Format: | GraphicsAddress[31:12] Specifies the lower bits of Graphics Address for the Start of the main surface where CCS has to be copied, if Destination Access Type is indirect access. If the Destination Access Type is indirect access this field indicates the CCS address in the native address space (actual virtual address within System Memory). It is 64K aligned if the access type is indirect access and 4K aligned if the access type is direct access.
	11:0	**Reserved** Access: | RO Format: | MBZ
4	31:25	**Destination MOCS** MOCS (Memory Object State Control) for destination operand.

XY_CTRL_SURF_COPY_BLT

		Programming Notes Destination MOCS value, which is used to program MOCS index for writing to memory, should select a MOCS register having "L3 Cacheability Control" programmed as uncacheable(UC) and "Global GO" parameter set as GOMemory (pushes GO point to memory). The MOCS Register may have L3 Lookup programmed as UCL3LKDIS for better efficiency.
	24:16	**Reserved** Format: MBZ
	15:0	**Destination Start Address High** Format: GraphicsAddress[47:32] Specifies the higher bits of Graphics Address for the Start of the main surface where CCS has to be copied, if the Destination Access type is indirect access. If the Destination Memory Access type is direct access this field indicates the higher address bits of the CCS address in the native address space.

XY_FAST_COLOR_BLT

<table>
<tr><th colspan="3">XY_FAST_COLOR_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3"><table><tr><th>Description</th></tr><tr><td>XY_FAST_COLOR_BLT instruction performs a color blit where the only operands involved are an in-line color source and destination surface of the same bit width. Legacy blit commands(2D BLT instructions other than XY_FAST_COPY_BLT, XY_BLOCK_COPY_BLT, XY_FAST_COLOR_BLT) and this fast color command can be interspersed. In case producer consumer relationship is detected between commands software must insert flush between them. Compression is supported by this new command.</td></tr></table></td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td><b>Client</b><table><tr><td>Default Value:</td><td>02h 2D Processor</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>28:22</td><td><b>Instruction Target(Opcode)</b><table><tr><td>Format:</td><td>Opcode</td></tr></table><table><tr><th>Value</th><th>Name</th></tr><tr><td>44h</td><td>XY_FAST_COLOR_BLT <b>[Default]</b></td></tr></table></td></tr>
<tr><td></td><td>21:19</td><td><b>Color Depth</b><table><tr><th>Description</th></tr><tr><td>This field actually programs bits per pixel value for each pixel of the surface. Reprogramming these bits require explicit flush of Copy Engine.</td></tr></table><table><tr><th>Value</th><th>Name</th></tr><tr><td>000b</td><td>8 bit color <b>[Default]</b></td></tr><tr><td>001b</td><td>16 bit color</td></tr><tr><td>010b</td><td>32 bit color</td></tr><tr><td>011b</td><td>64 bit color</td></tr><tr><td>100b</td><td>96 bit color (only supported for linear case)</td></tr><tr><td>101b</td><td>128 bit color</td></tr><tr><td>110b</td><td>RESERVED</td></tr><tr><td>111b</td><td>RESERVED</td></tr></table></td></tr>
<tr><td></td><td>18:14</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

XY_FAST_COLOR_BLT

13:12 **Special Mode of Operation**

This field indicates the mode of operation for the command.

Value	Name	Description
0h	NONE **[Default]**	No special mode. It will work as regular fill operation.
1h	FAST_CLEAR_1	Fast Clear writing 1's to CCS Buffer, which indicates clear state for the surface.
2h	FAST_CLEAR_0	Fast Clear to 0 during Clear Pass; Used to Initialize CCS Buffer with 0s to support Lossless Compressed Without Clear. This is possible only when Fast Clear is enabled.
3h	Reserved	Reserved for future use.

Programming Notes
AUX must be enabled in order to program "Special Operation mode" to FAST_CLEAR_0 or FAST_CLEAR_1 values. If FAST_CLEAR_0 or FAST_CLEAR_1 is enabled destination compression need not be enabled. FAST_CLEAR_0 and FAST_CLEAR_1 can only be programmed when Color Depth is not 96 BPP.

11:9 **Number of Multisamples**

This field indicates number of multi-samples on the surface.

Value	Name
000b	MULTISAMPLECOUNT_1 **[Default]**
001b	MULTISAMPLECOUNT_2
010b	MULTISAMPLECOUNT_4
011b	MULTISAMPLECOUNT_8
100b	MULTISAMPLECOUNT_16
101b	RESERVED
110b	RESERVED
111b	RESERVED

Programming Notes
Currently number of samples supported by copy engine is only MULTISAMPLECOUNT_1 . Rest of the values are for future use.

8 **Reserved**

Access:	RO
Format:	MBZ

7:0 **DWord Length**

Format:	=n

Total Length - 2

XY_FAST_COLOR_BLT

Value	Name
0Eh	Excludes DWORD 0,1 **[Default]**

1 31:30 **Destination Tiling**

These bits indicate destination tiling method.

Value	Name	Description	Programming Notes
00b	LINEAR **[Default]**	Linear mode (no tiling)	
01b	XMAJOR	X major tiling	
10b	Tile4	Tile4 4KB tiling	
11b	Tile64	Tile64 64KB tiling	Tile64 is not supported if surface type is 3D

29 **Destination Compression Enable**

Value	Name	Programming Notes
0b	Compression Disable **[Default]**	If compression is disabled AUX may or may not be enabled. When AUX is enabled (surface is compressible) "Special Operation Mode" can't be programmed as "NONE".
1b	Compression Enable	Compression enable require AUX to be enabled.

Programming Notes
Destination compression can be enabled irrespective of the value programmed in the destination target memory field (LOCAL_MEM or SYSTEM_MEM) as the value programmed in that field is considered as only performance hint.

28 **Destination Control Surface Type**

Value	Name
0b	3D control surface **[Default]**
1b	Media control surface

Programming Notes
If destination compression is enabled, control surface type can't be selected as media.

27:21 **Destination MOCS value**

MOCS (Memory Object State Control) for destination operand.

Programming Notes
Destination MOCS value, which is used to program MOCS index for writing to memory, should select a MOCS register having "L3 Cacheability Control" programmed as uncacheable(UC) and "Global GO" parameter set as GOMemory (pushes GO point to memory). The MOCS Register may have L3 Lookup programmed as UCL3LKDIS for better efficiency.

20:18 **Destination Auxiliary surface mode.**

Specifies type of the AUX surface associated with the primary surface (destination).

<table>
<tr><th colspan="3">XY_FAST_COLOR_BLT</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>000b</td><td>AUX_NONE <b>[Default]</b></td><td>No Auxiliary surface used,</td></tr>
<tr><td>001b</td><td>Reserved</td><td></td></tr>
<tr><td>010b</td><td>Reserved</td><td></td></tr>
<tr><td>011b</td><td>Reserved</td><td></td></tr>
<tr><td>100b</td><td>Reserved</td><td></td></tr>
<tr><td>101b</td><td>AUX_CCS_E</td><td>If Number of multisamples = 1, programming this value means lossless compression is enabled for that surface. Auxiliary surface is a CCS with linear tiling.</td></tr>
<tr><td>110b</td><td>Reserved</td><td></td></tr>
<tr><td>111b</td><td>Reserved</td><td></td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Only AUX_NONE and AUX_CCS_E values are valid now, rest of the values are reserved for future projects</td></tr>
</table>
</td></tr>
<tr><td></td><td>17:0</td><td><b>Destination Pitch</b>
<table>
<tr><td>Format:</td><td>U18-1</td></tr>
</table>
For <b>Linear Surfaces</b>, the pitch must be multiple of pixel width in bytes. For Tiled surfaces, the pitch has to be a <b>multiple of the Tile Width</b> (X direction width of the Tile) expressed in dwords (4 bytes).</td></tr>
<tr><td>2</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
The destination start line (inclusive) for Fast Color blit.
Format is 16-bit signed number.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>For Tiled case Y1 must be multiple of 4 when Fast Clear is enabled, if surface is not 1D.</td></tr>
</table>
</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
The destination start pixel (inclusive) for Fast Color blit.
Format is 16-bit signed number.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>The table below shows the programming restriction on X1 in terms of multiple of pixels when Fast Clear is enabled:
<table>
<tr><th>Color Depth</th><th>X1 is multiple of (Linear)</th><th>X1 is multiple of (Tiled)</th></tr>
<tr><td>8</td><td>128</td><td>32</td></tr>
<tr><td>16</td><td>64</td><td>16</td></tr>
<tr><td>32</td><td>32</td><td>8</td></tr>
<tr><td>64</td><td>16</td><td>4</td></tr>
<tr><td>128</td><td>8</td><td>2</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th colspan="3">XY_FAST_COLOR_BLT</th></tr>
<tr><td>3</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
The destination end line (exclusive) for Fast Color blit.
Format is 16-bit signed number.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>For Tiled case Y2 must be multiple of 4 when Fast Clear is enabled, if the surface is not 1D.</td></tr>
</table></td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X2 Coordinate (Right)</b>
The destination end pixel (exclusive) for Fast Color blit.
Format is 16-bit signed number.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>The table below shows the programming restriction on X2 in terms of multiple of pixels when Fast Clear is enabled:
<table>
<tr><th>Color Depth</th><th>X2 is multiple of (Linear)</th><th>X2 is multiple of (Tiled)</th></tr>
<tr><td>8</td><td>128</td><td>32</td></tr>
<tr><td>16</td><td>64</td><td>16</td></tr>
<tr><td>32</td><td>32</td><td>8</td></tr>
<tr><td>64</td><td>16</td><td>4</td></tr>
<tr><td>128</td><td>8</td><td>2</td></tr>
</table></td></tr>
</table></td></tr>
<tr><td>4..5</td><td>63:0</td><td><b>Destination Base Address</b>
<table>
<tr><td>Format:</td><td>GraphicsAddress[63:0]</td></tr>
</table>
<table>
<tr><th>Description</th></tr>
<tr><td>This bitfield contains the base address of the destination surface. When Tiling is enabled this address is cacheline (64Byte) aligned. When Destination Tiling is disabled, this address is byte aligned.</td></tr>
</table></td></tr>
<tr><td>6</td><td>31</td><td><b>Destination Target Memory</b>
Target memory for destination.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>LOCAL_MEM <b>[Default]</b></td><td>Target memory is local memory.</td></tr>
<tr><td>1b</td><td>SYSTEM_MEM</td><td>Target memory is system memory.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This field is used just as performance hint for destination target memory. If this bit is set as SYSTEM_MEM in the command, the writes will have target memory field set accordingly and those writes are given less priority in the system. When the number of such outstanding writes in the system crosses a certain threshold, copy engine is throttled. Please check registers 0x22200[15:2] (<b>BCS SW Control)</b> and 0x220A0[15:10] (<b>Mode Register for GAB</b>) for further details.</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

XY_FAST_COLOR_BLT

	29:16	**Destination Y offset** **Description** Format is U14. This field specifies the vertical offset in lines from the Surface Base Address to the start (origin) of the surface. **Programming Notes** For Linear surface Y offset must be 0. For Tiled surface Y offset must be greater than or equal to 0 and less than 16K in lines.
	15:14	**Reserved** Access: RO Format: MBZ
	13:0	**Destination X offset** **Description** Format is U14. This field specifies the horizontal offset in pixels from the Surface Base Address to the start (origin) of the surface. **Programming Notes** For Linear surface X offset must be 0. For Tiled surface X offset must be greater than or equal to 0 and less than 16K in pixels.
7..10	127:0	**Fill Color** **Description** The Dwords contains Color data to use for the fill operation. Format depends on the color depth selected and is always packed little endian way starting with DW[7][0]. 8 bit Color Format: DW[7][7:0] 16 bit Color Format: DW[7][15:0] 32 bit Color Format: DW[7][31:0] 64 bit Color Format: DW[8], DW[7] 96 bit Color Format: DW[9], DW[8], DW[7] 128 bit Color Format: DW[10], DW[9], DW[8], DW[7]
11	31:6	**Destination Clear Address Low** Format: GraphicsAddress[31:6] Specifies the lower bits of Graphics Address where clear value is stored in. The memory layout of the clear color pointed to by this address is a value stored in the lower-order bytes of the 64-byte cache-line.

<table>
<tr><th colspan="3">XY_FAST_COLOR_BLT</th></tr>
<tr><td></td><td>5</td><td><b>Destination Clear Value Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Programming Notes</th></tr>
<tr><td>0b</td><td>Disable <b>[Default]</b></td><td>[] If Special Mode of Operation is programmed to FAST_CLEAR_0 or FAST_CLEAR_1 (Fast Clear enabled) clear value must be disabled.</td></tr>
<tr><td>1b</td><td>Enable</td><td>Clear value can be enabled when Color Depth is not 96 BPP.</td></tr>
</table></td></tr>
<tr><td></td><td>4:0</td><td><b>Destination Compression Format</b>
This field indicates pixel or texel format of the surface used by compression/decompression logic.
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>Compression Format definition can be found in Structure_RENDER_SURFACE_STATE</td></tr>
<tr><td>For Unified Lossless Compression, Compression Format definition can be found Enumeration_RenderCompressionFormat</td></tr>
</table></td></tr>
<tr><td>12</td><td>31:16</td><td><b>Reserved</b>
<table>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>15:0</td><td><b>Destination Clear Address High</b>
<table>
<tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr>
</table>
Specifies the higher bits of Graphics Address of the destination surface where clear value is stored in the form of RGBA (R in the LSB and A in the MSB - in that order)</td></tr>
<tr><td>13</td><td>31:29</td><td><b>Destination Surface Type</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>SURFTYPE_1D <b>[Default]</b></td><td>Defines a 1-dimensional map or array of maps</td></tr>
<tr><td>1h</td><td>SURFTYPE_2D</td><td>Defines a 2-dimensional map or array of maps</td></tr>
<tr><td>2h</td><td>SURFTYPE_3D</td><td>Defines a 3-dimensional (volumetric) map</td></tr>
<tr><td>3h</td><td>SURFTYPE_CUBE</td><td>Defines cube maps or array of cube maps</td></tr>
<tr><td>4h</td><td>Reserved</td><td></td></tr>
<tr><td>5h</td><td>Reserved</td><td></td></tr>
<tr><td>6h</td><td>Reserved</td><td></td></tr>
<tr><td>7h</td><td>Reserved</td><td></td></tr>
</table></td></tr>
<tr><td></td><td>28</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>27:14</td><td><b>Destination Surface Width</b>
<table>
<tr><td>Format:</td><td>U14-1</td></tr>
</table>
This field specifies the width of the surface, minus 1. If the surface is MIP-mapped, this field specifies the width of the base MIP level. The width is specified in units of pixels or texels.</td></tr>
</table>

XY_FAST_COLOR_BLT

	13:0	**Destination Surface Height** Format: U14-1 This field specifies the height of the surface, minus 1. If the surface is MIP-mapped, this field contains the height of the base MIP level.
14	31:21	**Destination Surface Depth** Format: U11-1 This field specifies the total number of levels, minus 1, for a volume texture or the number of array elements, minus 1, allowed to be accessed starting at the Minimum Array Element for arrayed surfaces. If the volume texture is MIP-mapped, this field specifies the depth of the base MIP level.
	20:19	**Reserved** Format: MBZ
	18:4	**Destination Surface Qpitch** The interpretation of this field is dependent on Surface Type as follows: • SURFTYPE_1D: distance in *pixels* between array slices • SURFTYPE_2D/CUBE: distance in *rows* between array slices. For Quilted Textures this field specifies the distance in rows between *quilt* slices. For compressed texture formats, one row contains a complete compression block vertically. • SURFTYPE_3D: distance in *rows* between R-slices [**Note:** these *rows* are only in the vertical dimension without considering the depth dimension]. For compressed texture formats, one row contains a complete compression block vertically. • Other surface types: field is ignored **Value** | **Name** | **Description** [1h,7FFFh] | | in multiples of 4 (low 2 bits missing). The actual value of Qpitch is 4 times the value programmed.
	3:0	**Destination LOD** Default Value: [0h, Fh] LOD of the destination surface to be filled. **Programming Notes** Default value of destination LOD is 0. This value must be programmed as 0 for Fill operations involving non-MIP mapped surfaces.
15	31:21	**Destination Array Index** Format: U11-1 For arrayed surfaces (2D arrays, 1D arrays or cube arrays) this indicates the array index. For MIP MAPPED and volumetric surface this indicates the slice index of the destination surface to be copied.
	20:19	**Reserved** Access: RO Format: MBZ

XY_FAST_COLOR_BLT

	18	**Destination Depth/Stencil Resource** This bit field, when set, indicates if the resource is created as Depth/Stencil resource.
	17:12	**Reserved** Access: RO Format: MBZ
	11:8	**Destination Mip Tail Start LOD** This field indicates which LOD is the first one in the MIP tail if **Tiled Mode** is programmed to Tile64. The MIP tail has a different layout than the rest of the surface. Refer to the *Memory Data Formats* section for more details. **For other tiled formats and linear surfaces** this field is ignored.
	7:5	**Reserved** Format: MBZ
	4:3	**Destination Vertical Align** **Description** This field specifies the vertical alignment requirement in elements for destination surface. This field is used for 2D, CUBE, and 3D surface alignment when Tiled Mode is not Tile64. It is ignored for Tile64 surfaces. It is also ignored for 1D surfaces. See the appropriate Alignment table in the "Surface Layout and Tiling" section under Common Surface Formats for the table of alignment values for Tile64. More details about the field can be found in Structure_RENDER_SURFACE_STATE.
	2	**Reserved** Access: RO Format: MBZ
	1:0	**Destination Horizontal Align** This field specifies the horizontal alignment requirement for the surface. This field is ignored when **Tile Mode** is programmed to Tile64. See "Surface Layout and Tiling" section under Common Surface Formats for the table of alignment values for Tile64. Other details can be found in Structure_RENDER_SURFACE_STATE.

XY_FAST_COPY_BLT

<table>
<tr><th colspan="3">XY_FAST_COPY_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3"><b>Description</b>
This BLT instruction performs a color source copy where the only operands involved are a color source and destination of the same bit width. Note that this command does not support Clipping operations. This new blit command will happen in large numbers, consecutively, possibly an entire batch will comprise only new blit commands Legacy commands and new blit command will not be interspersed. If they are, they will be separated by implied HW flush: Whenever there is a transition between this new Fast Blit command and the Legacy Blit commands (2D BLT instructions other than XY_BLOCK_COPY_BLT, XY_FAST_COPY_BLT and XY_FAST_COLOR_BLT), the HW will impose an automatic flush BEFORE the execution (at the beginning) of the next blitter command. New blit command can use any combination of memory surface type - linear, tiledX, tiledY, and the tiling information is conveyed as part of the new Fast Copy command. The Fast Copy Blit supports the new 64KB Tiling. The starting pixel of Fast Copy blit for both source and destination should be on an OWord boundary.
Note that when two sequential fast copy blits have different source surfaces, but their destinations refer to the same destination surfaces and therefore destinations overlap it is imperative that a Flush be inserted between the two blits.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0
BR00</td><td>31:29</td><td><b>Client</b>
<table>
<tr><td>Default Value:</td><td>02h 2D Processor</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table>
<tr><td>Default Value:</td><td>42h</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>21:20</td><td><b>Source Tiling Method</b>
SW is required to flush the HW before changing the polarity of these bits for subsequent blits.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>00b</td><td>Linear (Tiling Disabled)</td><td></td></tr>
<tr><td>01b</td><td>TileX</td><td></td></tr>
<tr><td>10b</td><td>YMAJOR</td><td>Choosing between 'Legacy Tile-Y' or the 'Tile4' can be done in DWord 1, Bit[31].</td></tr>
<tr><td>11b</td><td>Tile64</td><td></td></tr>
</table></td></tr>
<tr><td></td><td>19:15</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

XY_FAST_COPY_BLT

14:13 **Destination Tiling Method**

SW is required to flush the HW before changing the polarity of these bits for subsequent blits.

Value	Name	Description
00b	Linear (Tiling Disabled)	
01b	TileX	
10b	YMAJOR	Choosing between 'Legacy Tile-Y' or the 'Tile4' can be done in DWord 1, Bit[30].
11b	Tile64	

12:8 **Reserved**

Access:	RO
Format:	MBZ

7:0 **DWord Length**

Default Value:	08h Excludes DWORD 0,1
Format:	=n

08h

1
BR13

31 **Tile Y Type for Source**

Source being Tile-Y can be selected in DWord 0, Bit[21:20].

Value	Name
1b	Tile4

30 **Tile Y Type for Destination**

Destination being Tile-Y can be selected in DWord 0, Bit[14:13].

Value	Name
1b	Tile4

29:28 **Reserved**

Access:	RO
Format:	MBZ

27 **Reserved**

Access:	RO
Format:	MBZ

26:24 **Color Depth**

Value	Name	Programming Notes
000b	8 bit color	
001b	16 bit color (565)	
010b	RESERVED	Programming of 010b is not supported.
011b	32 bit color	
100b	64 bit color (for 64KB Tiling)	

XY_FAST_COPY_BLT

		101b | 128 bit color (for 64KB Tiling)
	23:16	**Reserved**
	15:0	**Destination Pitch** Format: U16 **Description** For **Linear Surfaces**, the pitch must be multiple of pixel width in bytes. For **Tiled surfaces**, the pitch has to be a **multiple of the Tile width** (X direction width of the Tile). The number or value mentioned in this field here should be specified as a number in Dwords (4 byte quantity).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** Format: S15 Destination start line (inclusive) for Fast Copy blit.
	15:0	**Destination X1 Coordinate (Left)** Format: S15 Destination start pixel (inclusive) for Fast Copy blit. It should be on an OWord boundary.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** Format: S15 Destination end line (exclusive) for Fast Copy blit.
	15:0	**Destination X2 Coordinate (Right)** Format: S15 Destination end pixel (exclusive) for Fast Copy blit.
4..5	63:0	**Destination Base Address** Format: VIRTUAL_ADDR[63:0] This address must be Cache Line (64Byte) aligned for all surface types (linear or tiled).
6 BR26	31:16	**Source Y1 Coordinate (Top)** Format: S15 Source start line (inclusive) for Fast Copy blit.
	15:0	**Source X1 Coordinate (Left)** Source start pixel (inclusive) for Fast Copy blit. It should be on an OWord boundary.
7 BR11	31:16	**Reserved** Format: MBZ
	15:0	**Source Pitch** Format: U16 **Description** For **Linear Surfaces**, the pitch must be multiple of pixel width in bytes. For **Tiled surfaces**, the pitch has to be a **multiple of the Tile width** (X direction width of the Tile). The number or value mentioned in this field here should be specified as a number in

XY_FAST_COPY_BLT		
		Dwords (4 byte quantity).
8..9	63:0	**Source Base Address** Format: VIRTUAL_ADDR[63:0] This address must be Cache Line (64Byte) aligned for all surface types (linear or tiled).

XY_FULL_BLT

<table>
<tr><th colspan="3">XY_FULL_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source and pattern operands are the same bit width as the destination operand.
The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access. All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0
BR00</td><td>31:29</td><td><b>Client</b>
<table>
<tr><td>Default Value:</td><td>02h 2D Processor</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table>
<tr><td>Default Value:</td><td>55h</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>21:20</td><td><b>32bpp Byte Mask</b>
This field is only used for 32bpp.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td><b>[Default]</b></td></tr>
<tr><td>1xb</td><td>Write Alpha Channel</td></tr>
<tr><td>x1b</td><td>Write RGB Channel</td></tr>
</table></td></tr>
<tr><td></td><td>19:16</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>15</td><td><b>Src Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_FULL_BLT</th></tr>
<tr><td></td><td>14:12</td><td><b>Pattern Horizontal Seed</b>
Pixel of the scan line to start on corresponding to DST X=0.</td></tr>
<tr><td></td><td>11</td><td><b>Dest Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
<tr><td></td><td>10:8</td><td><b>Pattern Vertical Seed</b>
Starting scan line of the 8x8 pattern corresponding to DST Y=0.</td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Default Value:</td><td>0Ah</td></tr>
</table></td></tr>
<tr><td>1
BR13</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Clipping Enabled</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Disabled</td></tr>
<tr><td>1b</td><td>Enabled</td></tr>
</table></td></tr>
<tr><td></td><td>29:26</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>25:24</td><td><b>Color Depth</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>8 Bit Color</td></tr>
<tr><td>01b</td><td>16 Bit Color(565)</td></tr>
<tr><td>10b</td><td>16 Bit Color(1555)</td></tr>
<tr><td>11b</td><td>32 Bit Color</td></tr>
</table></td></tr>
<tr><td></td><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td></td><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement For Tiled surfaces (bit_15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be up to 128Kbytes (or 32KDwords).</td></tr>
<tr><td>2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
<tr><td>3
BR23</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X2 Coordinate (Right)</b>
16 bit signed number.</td></tr>
</table>

XY_FULL_BLT		
4..5	63:0	**Destination Base Address** Format: VIRTUAL_ADDR[63:0] Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address must be CL (64byte) aligned.
6 BR11	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Source Pitch (double word aligned and signed) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be up to 128Kbytes (or 32KDwords).
7 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
8..9	63:0	**Source Address** Format: VIRTUAL_ADDR[63:0] Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address must be CL (64byte) aligned.
10..11	63:0	**Pattern Base Address** Format: VIRTUAL_ADDR[63:0] Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address must be CL (64byte) aligned.

XY_FULL_IMMEDIATE_PATTERN_BLT

XY_FULL_IMMEDIATE_PATTERN_BLT

Source:	BlitterCS
Length Bias:	2

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source and immediate pattern operands are the same bit width as the destination operand. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64 DWs) for 8, 16, and 32 bpp color patterns. DWL indicates the total number of Dwords of immediate data.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 74h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value | Name 00b | **[Default]** 1xb | Write Alpha Channel x1b | Write RGB Channel
	19:16	**Reserved** Access: RO Format: MBZ
	15	**Src Tiling Enable** Value | Name | Description 0b | Tiling Disabled (Linear) | 1b | Tiling Enabled | : Tile-X or Tile-Y.

<table>
<tr><th colspan="3">XY_FULL_IMMEDIATE_PATTERN_BLT</th></tr>
<tr><td></td><td>14:12</td><td><b>Pattern Horizontal Seed</b>
(pixel of the scan line to start on corresponding to DST X=0)</td></tr>
<tr><td></td><td>11</td><td><b>Dest Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
<tr><td></td><td>10:8</td><td><b>Pattern Vertical Seed</b>
Starting scan line of the 8x8 pattern corresponding to DST Y=0.</td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Format:</td><td>=n</td></tr>
</table>
n = 08 + DWL = (where 'DWL' is Number of Immediate data in terms of double words). Immediate data size is 16 DW for 8 BPP, 32 DW for 16 BPP and 64 DW for 32 BPP.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>[24,72]</td><td>Excludes DWORD 0,1</td></tr>
</table></td></tr>
<tr><td>1
BR13</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Clipping Enabled</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Disabled</td></tr>
<tr><td>1b</td><td>Enabled</td></tr>
</table></td></tr>
<tr><td></td><td>29:26</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>25:24</td><td><b>Color Depth</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>8 Bit Color</td></tr>
<tr><td>01b</td><td>16 Bit Color(565)</td></tr>
<tr><td>10b</td><td>16 Bit Color(1555)</td></tr>
<tr><td>11b</td><td>32 Bit Color</td></tr>
</table></td></tr>
<tr><td></td><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td></td><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement For Tiled surfaces (bit_15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).</td></tr>
<tr><td>2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
</table>

XY_FULL_IMMEDIATE_PATTERN_BLT		
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5	63:0	**Destination Base Address** Format: VIRTUAL_ADDR[63:0] Base address of the destination surface: X=0, Y=0. When Src Tiling is enabled (Bit_15 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
6 BR11	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Source Pitch (double word aligned and signed) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be up to 128Kbytes (or 32KDwords).
7 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
8..9	63:0	**Source Address** Format: VIRTUAL_ADDR[63:0] Base address of the source surface: X=0, Y=0. When Src Tiling is enabled (Bit_15 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
10..n	31:0	**Immediate Data 0**

XY_FULL_MONO_PATTERN_BLT

XY_FULL_MONO_PATTERN_BLT

Source:	BlitterCS
Length Bias:	2

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The pattern operand is monochrome and the source operand is the same bit width as the destination operand.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the pattern foreground color is used in the ROP operation.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

Setting both Solid Pattern Select =1 and Mono Pattern Transparency = 1 is mutually exclusive. The device implementation results in NO PIXELs DRAWN.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="4">0
BR00</td><td>31:29</td><td><b>Client</b>
<table>
<tr><td>Default Value:</td><td>02h 2D Processor</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table>
<tr><td>Default Value:</td><td>57h</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td>21:20</td><td><b>32bpp Byte Mask</b>
This field is only used for 32bpp.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td><b>[Default]</b></td></tr>
<tr><td>1xb</td><td>Write Alpha Channel</td></tr>
<tr><td>x1b</td><td>Write RGB Channel</td></tr>
</table></td></tr>
<tr><td>19:16</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_FULL_MONO_PATTERN_BLT</th></tr>
<tr><td></td><td>15</td><td><b>Src Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
<tr><td></td><td>14:12</td><td><b>Pattern Horizontal Seed</b>
(pixel of the scan line to start on corresponding to DST X=0)</td></tr>
<tr><td></td><td>11</td><td><b>Dest Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
<tr><td></td><td>10:8</td><td><b>Pattern Vectical Seed</b>
Starting scan line of the 8x8 pattern corresponding to DST Y=0.</td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0Ch</td><td></td></tr>
</table></td></tr>
<tr><td>1
BR13</td><td>31</td><td><b>Solid Pattern Select</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>No Solid Pattern</td></tr>
<tr><td>1</td><td>Solid Pattern</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Clipping Enabled</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Disabled</td></tr>
<tr><td>1b</td><td>Enabled</td></tr>
</table></td></tr>
<tr><td></td><td>29</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>28:27</td><td><b>Mono Source Transparency Mode</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Use Background</td></tr>
<tr><td>1</td><td>Transparency Enabled</td></tr>
</table></td></tr>
<tr><td></td><td>26</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_FULL_MONO_PATTERN_BLT</th></tr>
<tr><td></td><td>25:24</td><td><b>Color Depth</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>8 Bit Color</td></tr>
<tr><td>01b</td><td>16 Bit Color(565)</td></tr>
<tr><td>10b</td><td>16 Bit Color(1555)</td></tr>
<tr><td>11b</td><td>32 Bit Color</td></tr>
</table></td></tr>
<tr><td></td><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td></td><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be up to 128Kbytes (or 32KDwords).</td></tr>
<tr><td>2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
<tr><td>3
BR23</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X2 Coordinate (Right)</b>
16 bit signed number.</td></tr>
<tr><td>4..5</td><td>63:0</td><td><b>Destination Base Address</b>
<table>
<tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr>
</table>
Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_15 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.</td></tr>
<tr><td>6
BR11</td><td>31:16</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>15:0</td><td><b>Source Pitch (double word aligned and signed) and in DWords</b>
2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be up to 128Kbytes (or 32KDwords).</td></tr>
<tr><td>7
BR26</td><td>31:16</td><td><b>Source Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Source X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
<tr><td>8..9</td><td>63:0</td><td><b>Source Address</b>
<table>
<tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr>
</table>
Base address of the source surface: X=0, Y=0. When Src Tiling is enabled (Bit_15 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.</td></tr>
<tr><td>10
BR16</td><td>31:0</td><td><b>Pattern Background Color</b>
8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]</td></tr>
</table>

XY_FULL_MONO_PATTERN_BLT		
11 BR17	31:0	**Pattern Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
12 BR20	31:0	**Pattern Data 0** (least significant DW)
13 BR21	31:0	**Pattern Data 1** (most significant DW)

XY_FULL_MONO_PATTERN_MONO_SRC_BLT

<table>
<tr><th colspan="3">XY_FULL_MONO_PATTERN_MONO_SRC_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">The full BLT provides the ability to specify all 3 operands: destination, source, and pattern. The pattern and source operands are monochrome.
The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation.
All non-text monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the destination X1 coordinate.
The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the source bit is 1, then the pattern foreground color is used in the ROP operation. The monochrome source transparency mode works identical to the pattern transparency mode.
All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.
Setting both Solid Pattern Select =1 and Mono Pattern Transparency = 1 is mutually exclusive. The device implementation results in NO PIXELs DRAWN.
Negative Stride (= Pitch) is NOT ALLOWED.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="4">0
BR00</td><td>31:29</td><td><b>Client</b><table><tr><td>Default Value:</td><td>02h 2D Processor</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td>28:22</td><td><b>Instruction Target(Opcode)</b><table><tr><td>Default Value:</td><td>58h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td>21:20</td><td><b>32bpp Byte Mask</b>
This field is only used for 32bpp.<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td><b>[Default]</b></td></tr><tr><td>1xb</td><td>Write Alpha Channel</td></tr><tr><td>x1b</td><td>Write RGB Channel</td></tr></table></td></tr>
<tr><td>19:17</td><td><b>Monochrome source data bit position of the first pixel within a byte per scan line.</b></td></tr>
</table>

<table>
<tr><th colspan="3">XY_FULL_MONO_PATTERN_MONO_SRC_BLT</th></tr>
<tr><td></td><td>16:15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>14:12</td><td><b>Pattern Horizontal Seed</b>
(pixel of the scan line to start on corresponding to DST X=0)</td></tr>
<tr><td></td><td>11</td><td><b>Tiling Enable</b><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr><tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr></table></td></tr>
<tr><td></td><td>10:8</td><td><b>Pattern Vertical Seed</b>
Starting scan line of the 8x8 pattern corresponding to DST Y = 0.</td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0Ch</td><td></td></tr></table></td></tr>
<tr><td>1
BR13</td><td>31</td><td><b>Solid Pattern Select</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>No Solid Pattern</td></tr><tr><td>1</td><td>Solid Pattern</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Clipping Enabled</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0b</td><td>Disabled</td></tr><tr><td>1b</td><td>Enabled</td></tr></table></td></tr>
<tr><td></td><td>29</td><td><b>Mono Source Transparency Mode</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Use Background</td></tr><tr><td>1</td><td>Transparency Enabled</td></tr></table></td></tr>
<tr><td></td><td>28</td><td><b>Mono Pattern Transparency Mode</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Use Background</td></tr><tr><td>1</td><td>Transparency Enabled</td></tr></table></td></tr>
<tr><td></td><td>27:26</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_FULL_MONO_PATTERN_MONO_SRC_BLT</th></tr>
<tr><td rowspan="3"></td><td>25:24</td><td><b>Color Depth</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>8 Bit Color</td></tr>
<tr><td>01b</td><td>16 Bit Color(565)</td></tr>
<tr><td>10b</td><td>16 Bit Color(1555)</td></tr>
<tr><td>11b</td><td>32 Bit Color</td></tr>
</table></td></tr>
<tr><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).</td></tr>
<tr><td rowspan="2">2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
<tr><td rowspan="2">3
BR23</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
16 bit signed number.</td></tr>
<tr><td>15:0</td><td><b>Destination X2 Coordinate (Right)</b>
16 bit signed number.</td></tr>
<tr><td>4..5
This bitfield contains the base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.</td><td>63:0</td><td><b>Destination Base Address</b>
<table>
<tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr>
</table></td></tr>
<tr><td>6..7
Address corresponds to DST X1, Y1. Note no NPO2 change here.
The Mono Source Base Address must always be Cache Line (64byte) aligned.</td><td>63:0</td><td><b>Mono Source Address</b>
<table>
<tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr>
</table></td></tr>
<tr><td>8
BR18</td><td>31:0</td><td><b>Source Background Color</b>
8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]</td></tr>
<tr><td>9
BR19</td><td>31:0</td><td><b>Source Foreground Color</b>
8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]</td></tr>
<tr><td>10
BR16</td><td>31:0</td><td><b>Pattern Background Color</b>
8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]</td></tr>
<tr><td>11
BR17</td><td>31:0</td><td><b>Pattern Foreground Color</b>
8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]</td></tr>
<tr><td>12
BR20</td><td>31:0</td><td><b>Pattern Data 0</b>
(least significant DW)</td></tr>
</table>

XY_FULL_MONO_PATTERN_MONO_SRC_BLT		
13 BR21	31:0	**Pattern Data 1** (most significant DW)

XY_FULL_MONO_SRC_BLT

<table>
<tr><th colspan="3">XY_FULL_MONO_SRC_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source operand is monochrome and the pattern operand is the same bit width as the destination.
The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation.
All non-text and non-immediate monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the Destination X1 coordinate.
All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.
Negative Stride (= Pitch) is NOT ALLOWED</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="6">0
BR00</td><td>31:29</td><td><b>Client</b>
<table>
<tr><td>Default Value:</td><td>02h 2D Processor</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table>
<tr><td>Default Value:</td><td>56h</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td>21:20</td><td><b>32bpp Byte Mask</b>
This field is only used for 32bpp.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td><b>[Default]</b></td></tr>
<tr><td>1xb</td><td>Write Alpha Channel</td></tr>
<tr><td>x1b</td><td>Write RGB Channel</td></tr>
</table></td></tr>
<tr><td>19:17</td><td><b>Monochrome source data bit position of the first pixel within a byte per scan line.</b></td></tr>
<tr><td>16:15</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>14:12</td><td><b>Pattern Horizontal Seed</b>
(pixel of the scan line to start on corresponding to DST X=0)</td></tr>
</table>

<table>
<tr><th colspan="3">XY_FULL_MONO_SRC_BLT</th></tr>
<tr><td rowspan="3"></td><td>11</td><td><b>Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
<tr><td>10:8</td><td><b>Pattern Vertical Seed</b>
Starting scan line of the 8x8 pattern corresponding to DST Y = 0.</td></tr>
<tr><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0Ah</td><td></td></tr>
</table></td></tr>
<tr><td rowspan="8">1
BR13</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>30</td><td><b>Clipping Enabled</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Disabled</td></tr>
<tr><td>1b</td><td>Enabled</td></tr>
</table></td></tr>
<tr><td>29</td><td><b>Mono Source Transparency Mode</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Use Background</td></tr>
<tr><td>1</td><td>Transparency Enabled</td></tr>
</table></td></tr>
<tr><td>28:26</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>25:24</td><td><b>Color Depth</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>8 Bit Color</td></tr>
<tr><td>01b</td><td>16 Bit Color(565)</td></tr>
<tr><td>10b</td><td>16 Bit Color(1555)</td></tr>
<tr><td>11b</td><td>32 Bit Color</td></tr>
</table></td></tr>
<tr><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).</td></tr>
<tr><td>2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
</table>

XY_FULL_MONO_SRC_BLT

	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address must be CL (64byte) aligned.	63:0	**Destination Base Address** Format: VIRTUAL_ADDR[63:0]
6..7 Address corresponds to DST X1, Y1. Note no NPO2 change here. The Mono Source Base Address must always be Cache Line (64byte) aligned.	63:0	**Mono Source Address** Format: VIRTUAL_ADDR[63:0]
8 BR18	31:0	**Source Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
9 BR19	31:0	**Source Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
10..11 28:06 are implemented. Note no NPO2 change here. The pattern data must be located in linear memory. The programmed Pattern Base Address must always be Cache Line (64byte) aligned.	63:0	**Pattern Base Address** Format: VIRTUAL_ADDR[63:0]

XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT

XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT

Source:	BlitterCS
Length Bias:	2

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source operand is a monochrome and the immediate pattern operand is the same bit width as the destination. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns. The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation. All non-text monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the destination X1 coordinate. All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation. The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8. Negative Stride (= Pitch) is NOT ALLOWED.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="6">0
BR00</td><td>31:29</td><td><b>Client</b><table><tr><td>Default Value:</td><td>02h 2D Processor</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td>28:22</td><td><b>Instruction Target(Opcode)</b><table><tr><td>Default Value:</td><td>75h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td>21:20</td><td><b>32bpp Byte Mask</b>
This field is only used for 32bpp.<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td><b>[Default]</b></td></tr><tr><td>1xb</td><td>Write Alpha Channel</td></tr><tr><td>x1b</td><td>Write RGB Channel</td></tr></table></td></tr>
<tr><td>19:17</td><td><b>Monochrome source data bit position of the first pixel within a byte per scan line.</b></td></tr>
<tr><td>16:15</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>14:12</td><td><b>Pattern Horizontal Seed</b>
(pixel of the scan line to start on corresponding to DST X=0)</td></tr>
</table>

<table>
<tr><th colspan="3">XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT</th></tr>
<tr><td></td><td>11</td><td><b>Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
<tr><td></td><td>10:8</td><td><b>Pattern Vertical Seed</b>
Starting scan line of the 8x8 pattern corresponding to DST Y=0.</td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Format:</td><td>=n</td></tr>
</table>
n = 08 + DWL , (where 'DWL' is Number of Immediate data in terms of double words). Immediate data size is 16 DW for 8 BPP, 32 DW for 16 BPP and 64 DW for 32 BPP.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>[24,72]</td><td>Excludes DWORD 0,1</td></tr>
</table></td></tr>
<tr><td>1
BR13</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Clipping Enabled</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Disabled</td></tr>
<tr><td>1b</td><td>Enabled</td></tr>
</table></td></tr>
<tr><td></td><td>29</td><td><b>Mono Source Transparency Mode</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Use Background</td></tr>
<tr><td>1</td><td>Transparency Enabled</td></tr>
</table></td></tr>
<tr><td></td><td>28:26</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>25:24</td><td><b>Color Depth</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>8 Bit Color</td></tr>
<tr><td>01b</td><td>16 Bit Color(565)</td></tr>
<tr><td>10b</td><td>16 Bit Color(1555)</td></tr>
<tr><td>11b</td><td>32 Bit Color</td></tr>
</table></td></tr>
<tr><td></td><td>23:16</td><td><b>Raster Operation</b></td></tr>
</table>

<table>
<tr><th colspan="3">XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT</th></tr>
<tr><td></td><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be up to 128Kbytes (or 32KDwords).</td></tr>
<tr><td rowspan="2">2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
<tr><td rowspan="2">3
BR23</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
16 bit signed number.</td></tr>
<tr><td>15:0</td><td><b>Destination X2 Coordinate (Right)</b>
16 bit signed number.</td></tr>
<tr><td>4..5
Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.</td><td>63:0</td><td><b>Destination Base Address</b><table><tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr></table></td></tr>
<tr><td>6..7
Address corresponds to DST X1, Y1. Note no NPO2 change here. The Mono Source Base Address must always be Cache Line (64byte) aligned.</td><td>63:0</td><td><b>Mono Source Address</b><table><tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr></table></td></tr>
<tr><td>8
BR18</td><td>31:0</td><td><b>Source Background Color</b>
8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]</td></tr>
<tr><td>9
BR19</td><td>31:0</td><td><b>Source Foreground Color</b>
8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]</td></tr>
<tr><td>10..n</td><td>31:0</td><td><b>Immediate Data</b></td></tr>
</table>

XY_MONO_PAT_BLT

<table>
<tr><th colspan="3">XY_MONO_PAT_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">MONO_PAT_BLT is used when we have no source and the monochrome pattern is not trivial (is not a solid color only). The monochrome pattern is loaded from the instruction stream.
All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.
The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the pattern bit is 1, then the pattern foreground color is used in the ROP operation.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="6">0
BR00</td><td>31:29</td><td><b>Client</b>
<table>
<tr><td>Default Value:</td><td>02h 2D Processor</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table>
<tr><td>Default Value:</td><td>52h</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td>21:20</td><td><b>32bpp Byte Mask</b>
This field is only used for 32bpp.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td><b>[Default]</b></td></tr>
<tr><td>1xb</td><td>Write Alpha Channel</td></tr>
<tr><td>x1b</td><td>Write RGB Channel</td></tr>
</table></td></tr>
<tr><td>19:15</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>14:12</td><td><b>Pattern Horizontal Seed</b>
Pixel of the scan line to start on corresponding to DST X=0.</td></tr>
<tr><td>11</td><td><b>Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>Tile-X or Tile-Y.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_MONO_PAT_BLT</th></tr>
<tr><td rowspan="2"></td><td>10:8</td><td><b>Pattern Vertical Seed</b>
Scan line of the 8x8 pattern to start on corresponding to DST Y=0.</td></tr>
<tr><td>7:0</td><td><b>DWord Length</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>08h</td><td></td></tr></table></td></tr>
<tr><td rowspan="8">1
BR13</td><td>31</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>30</td><td><b>Clipping Enabled</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0b</td><td>Disabled</td></tr><tr><td>1b</td><td>Enabled</td></tr></table></td></tr>
<tr><td>29</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>28</td><td><b>Mono Pattern Transparency Mode</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Use Background</td></tr><tr><td>1</td><td>Transparency Enabled</td></tr></table></td></tr>
<tr><td>27:26</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>25:24</td><td><b>Color Depth</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>8 Bit Color</td></tr><tr><td>01b</td><td>16 Bit Color(565)</td></tr><tr><td>10b</td><td>16 Bit Color(1555)</td></tr><tr><td>11b</td><td>32 Bit Color</td></tr></table></td></tr>
<tr><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).</td></tr>
<tr><td rowspan="2">2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
</table>

XY_MONO_PAT_BLT		
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: VIRTUAL_ADDR[63:0]
6 BR16	31:0	**Pattern Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 BR17	31:0	**Pattern Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
8 BR20	31:0	**Pattern Data 0**
9 BR21	31:0	**Pattern Data 1**

XY_MONO_PAT_FIXED_BLT

XY_MONO_PAT_FIXED_BLT	
Source:	BlitterCS
Length Bias:	2

MONO_PAT_FIXED_BLT is used when we have no source and the monochrome pattern is not trivial (is not a solid color only). The monochrome pattern is one of 10 fixed patterns described below. The pattern seeds can still be used with the fixed patterns, creating even more fixed patterns. This eliminates 2 doublewords compared to the XY_MONO_PAT_BLT command packet.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the pattern bit is 1, then the pattern foreground color is used in the ROP operation.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 59h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value / Name: 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19	**Reserved** Access: RO Format: MBZ
	18:15	**Fixed Pattern** Value / Name: 0000b HS_HORIZONTAL 0001b HS_VERTICAL 0010b HS_FDIAGONAL 0011b HS_BDIAGONAL 0100b HS_CROSS

<table>
<tr><th colspan="3">XY_MONO_PAT_FIXED_BLT</th></tr>
<tr><td></td><td></td><td><table>
<tr><td>0101b</td><td>HS_DIAGCROSS</td></tr>
<tr><td>0110b</td><td>Reserved</td></tr>
<tr><td>0111b</td><td>Reserved</td></tr>
<tr><td>1000b</td><td>Screen Door</td></tr>
<tr><td>1001b</td><td>SD Wide</td></tr>
<tr><td>1010b</td><td>Walking Bit (one)</td></tr>
<tr><td>1011b</td><td>Walking Zero</td></tr>
<tr><td>1100b</td><td>Reserved</td></tr>
<tr><td>1101b</td><td>Reserved</td></tr>
<tr><td>1110b</td><td>Reserved</td></tr>
<tr><td>1111b</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td></td><td>14:12</td><td><b>Pattern Horizontal Seed</b>
Pixel of the scan line to start on corresponding to DST X=0.</td></tr>
<tr><td></td><td>11</td><td><b>Tiling Enable</b><table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
<tr><td></td><td>10:8</td><td><b>Pattern Vertical Seed</b>
Scan line of the 8x8 pattern to start on corresponding to DST Y=0.</td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b><table>
<tr><td>Format:</td><td>=n</td></tr>
</table><table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>06h</td><td></td></tr>
</table></td></tr>
<tr><td>1
BR13</td><td>31</td><td><b>Reserved</b><table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Clipping Enabled</b><table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Disabled</td></tr>
<tr><td>1b</td><td>Enabled</td></tr>
</table></td></tr>
<tr><td></td><td>29</td><td><b>Reserved</b><table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_MONO_PAT_FIXED_BLT</th></tr>
<tr><td></td><td>28</td><td><b>Mono Pattern Transparency Mode</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0</td><td>Use Background</td></tr>
<tr><td>1</td><td>Transparency Enabled</td></tr>
</table></td></tr>
<tr><td></td><td>27:26</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>25:24</td><td><b>Color Depth</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>8 Bit Color</td></tr>
<tr><td>01b</td><td>16 Bit Color(565)</td></tr>
<tr><td>10b</td><td>16 Bit Color(1555)</td></tr>
<tr><td>11b</td><td>32 Bit Color</td></tr>
</table></td></tr>
<tr><td></td><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td></td><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).</td></tr>
<tr><td>2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
<tr><td>3
BR23</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X2 Coordinate (Right)</b>
16 bit signed number.</td></tr>
<tr><td>4..5</td><td>63:0</td><td><b>Destination Base Address</b>
<table>
<tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr>
</table>
Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.</td></tr>
<tr><td>6
BR16</td><td>31:0</td><td><b>Pattern Background Color</b>
8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]</td></tr>
<tr><td>7
BR17</td><td>31:0</td><td><b>Pattern Foreground Color</b>
8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]</td></tr>
</table>

XY_MONO_SRC_COPY_BLT

<table>
<tr><th colspan="3">XY_MONO_SRC_COPY_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">This BLT instruction performs a monochrome source copy where the only operands involved is a monochrome source and destination. The source and destination operands cannot overlap therefore the X and Y directions are always forward.
All non-text monochrome sources are word aligned. At the end of a scan line of monochrome source, all bits until the next word boundary are ignored. The monochrome source data bit position field [2:0] indicates the bit position within the first byte of the scan line that should be used as the first source pixel which corresponds to the destination X1 coordinate.
The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation. The ROP value chosen must involve source and no pattern data in the ROP operation.
Negative Stride (= Pitch) is NOT ALLOWED.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0
BR00</td><td>31:29</td><td><b>Client</b>
<table>
<tr><td>Default Value:</td><td>02h 2D Processor</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table>
<tr><td>Default Value:</td><td>54h</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>21:20</td><td><b>32bpp Byte Mask</b>
This field is only used for 32bpp.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td><b>[Default]</b></td></tr>
<tr><td>1xb</td><td>Write Alpha Channel</td></tr>
<tr><td>x1b</td><td>Write RGB Channel</td></tr>
</table></td></tr>
<tr><td></td><td>19:17</td><td><b>Monochrome source data bit position of the first pixel within a byte per scan line.</b></td></tr>
<tr><td></td><td>16:12</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>11</td><td><b>Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_MONO_SRC_COPY_BLT</th></tr>
<tr><td></td><td>10:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>08h</td><td></td></tr></table></td></tr>
<tr><td>1
BR13</td><td>31</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Clipping Enabled</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0b</td><td>Disabled</td></tr><tr><td>1b</td><td>Enabled</td></tr></table></td></tr>
<tr><td></td><td>29</td><td><b>Mono Source Transparency Mode</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Use Background</td></tr><tr><td>1</td><td>Transparency Enabled</td></tr></table></td></tr>
<tr><td></td><td>28:26</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>25:24</td><td><b>Color Depth</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>8 Bit Color</td></tr><tr><td>01b</td><td>16 Bit Color(565)</td></tr><tr><td>10b</td><td>16 Bit Color(1555)</td></tr><tr><td>11b</td><td>32 Bit Color</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td></td><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).</td></tr>
<tr><td>2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
<tr><td>3
BR23</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
16 bit signed number.</td></tr>
</table>

XY_MONO_SRC_COPY_BLT		
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: | VIRTUAL_ADDR[63:0]
6..7 Address corresponds to DST X1, Y1. Note no NPO2 change here. The Mono Source Base Address must always be Cache Line (64byte) aligned.	63:0	**Mono Source Address** Format: | VIRTUAL_ADDR[63:0]
8 BR18	31:0	**Source Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
9 BR19	31:0	**Source Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

XY_MONO_SRC_COPY_IMMEDIATE_BLT

XY_MONO_SRC_COPY_IMMEDIATE_BLT

Source:	BlitterCS
Length Bias:	2

This instruction allows the Driver to send monochrome data through the instruction stream, eliminating the read latency of the source during command execution.
The IMMEDIATE_BLT data MUST transfer an even number of doublewords and the exact number of quadwords. DWL indicates the total number of Dwords of immediate data.
All non-text monochrome sources are word aligned. At the end of a scan line of monochrome source, all bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates the bit position within the first byte of the scan line that should be used as the first source pixel which corresponds to the destination X1 coordinate.
The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation. The ROP value chosen must involve source and no pattern data in the ROP operation.
The monochrome source data supplied corresponds to the Destination X1 and Y1 coordinates.
Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 71h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value / Name: 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:12	**Reserved** Access: RO Format: MBZ
	11	**Tiling Enable** Value / Name / Description: 0b Tiling Disabled (Linear) 1b Tiling Enabled : Tile-X or Tile-Y.

<table>
<tr><th colspan="3">XY_MONO_SRC_COPY_IMMEDIATE_BLT</th></tr>
<tr><td></td><td>10:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b><table><tr><td>Format:</td><td>=n</td></tr></table>n = 06 + DWL
Where DWL is number of immediate data in terms of dwords. Immediate data size is 16 DW for 8 BPP, 32 DW for 16 BPP and 64 DW for 32 BPP.<table><tr><th>Value</th><th>Name</th></tr><tr><td>[22,70]</td><td>Excludes DWORD 0,1</td></tr></table></td></tr>
<tr><td>1
BR13</td><td>31</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Clipping Enabled</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0b</td><td>Disabled</td></tr><tr><td>1b</td><td>Enabled</td></tr></table></td></tr>
<tr><td></td><td>29</td><td><b>Mono Source Transparency Mode</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0b</td><td>Transparency Enabled</td></tr><tr><td>1b</td><td>Use Background</td></tr></table></td></tr>
<tr><td></td><td>28:26</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>25:24</td><td><b>Color Depth</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>8 Bit Color</td></tr><tr><td>01b</td><td>16 Bit Color(565)</td></tr><tr><td>10b</td><td>16 Bit Color(1555)</td></tr><tr><td>11b</td><td>32 Bit Color</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td></td><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).</td></tr>
<tr><td>2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
</table>

XY_MONO_SRC_COPY_IMMEDIATE_BLT		
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: VIRTUAL_ADDR[63:0]
6 BR18	31:0	**Source Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 BR19	31:0	**Source Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
8..n	31:0	**Immediate Data**

XY_PAT_BLT_IMMEDIATE

XY_PAT_BLT_IMMEDIATE

Source:	BlitterCS
Length Bias:	2

PAT_BLT_IMMEDIATE is used when there is no source and the color pattern is not trivial (is not a solid color only) and the pattern is pulled through the command stream. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns.
DWL indicates the total number of Dwords of immediate data. All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 72h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value / Name: 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:15	**Reserved** Access: RO Format: MBZ
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** Value / Name / Description: 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled : Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.

<table>
<tr><th colspan="3">XY_PAT_BLT_IMMEDIATE</th></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Default Value:</td><td>[20,68] Excludes DWORD 0,1</td></tr>
<tr><td>Format:</td><td>=n</td></tr>
</table>
n = 04 + DWL
Where DWL indicates number of immediate data in terms of dwords.</td></tr>
<tr><td>1
BR13</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Clipping Enabled</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Disabled</td></tr>
<tr><td>1b</td><td>Enabled</td></tr>
</table></td></tr>
<tr><td></td><td>29:26</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>25:24</td><td><b>Color Depth</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>8 Bit Color</td></tr>
<tr><td>01b</td><td>16 Bit Color(565)</td></tr>
<tr><td>10b</td><td>16 Bit Color(1555)</td></tr>
<tr><td>11b</td><td>32 Bit Color</td></tr>
</table></td></tr>
<tr><td></td><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td></td><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).</td></tr>
<tr><td>2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
<tr><td>3
BR23</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X2 Coordinate (Right)</b>
16 bit signed number.</td></tr>
</table>

XY_PAT_BLT_IMMEDIATE		
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: VIRTUAL_ADDR[63:0]
6..n	31:0	**Immediate Data**

XY_PAT_BLT

<table>
<tr><th colspan="3">XY_PAT_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">PAT_BLT is used when there is no source and the color pattern is not trivial (is not a solid color only).
If clipping is enabled, all scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation. The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="8">0
BR00</td><td>31:29</td><td><b>Client</b>
<table><tr><td>Default Value:</td><td>02h 2D Processor</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table><tr><td>Default Value:</td><td>51h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td>21:20</td><td><b>32bpp Byte Mask</b>
This field is only used for 32bpp.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td><b>[Default]</b></td></tr><tr><td>1xb</td><td>Write Alpha Channel</td></tr><tr><td>x1b</td><td>Write RGB Channel</td></tr></table></td></tr>
<tr><td>19:15</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>14:12</td><td><b>Pattern Horizontal Seed</b>
Pixel of the scan line to start on corresponding to DST X=0.</td></tr>
<tr><td>11</td><td><b>Tiling Enable</b>
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr><tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr></table></td></tr>
<tr><td>10:8</td><td><b>Pattern Vertical Seed</b>
Scan line of the 8x8 pattern to start on corresponding to DST Y=0.</td></tr>
<tr><td>7:0</td><td><b>DWord Length</b>
<table><tr><td>Default Value:</td><td>06h</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_PAT_BLT</th></tr>
<tr><td rowspan="6">1
BR13</td><td>31</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>30</td><td><b>Clipping Enabled</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0b</td><td>Disabled</td></tr><tr><td>1b</td><td>Enabled</td></tr></table></td></tr>
<tr><td>29:26</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>25:24</td><td><b>Color Depth</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>8 Bit Color</td></tr><tr><td>01b</td><td>16 Bit Color(565)</td></tr><tr><td>10b</td><td>16 Bit Color(1555)</td></tr><tr><td>11b</td><td>32 Bit Color</td></tr></table></td></tr>
<tr><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be up to 128Kbytes (or 32KDwords).</td></tr>
<tr><td rowspan="2">2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
<tr><td rowspan="2">3
BR23</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
16 bit signed number.</td></tr>
<tr><td>15:0</td><td><b>Destination X2 Coordinate (Right)</b>
16 bit signed number.</td></tr>
<tr><td>4..5
Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.</td><td>63:0</td><td><b>Destination Base Address</b><table><tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_PAT_BLT</th></tr>
<tr><td>6..7
28:06 are implemented. Note no NPO2 change here. The pattern data must be located in linear memory. The programmed Pattern Base Address must always be Cache Line (64byte) aligned.</td><td>63:0</td><td><b>Pattern Base Address</b><table><tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr></table></td></tr>
</table>

XY_PAT_CHROMA_BLT_IMMEDIATE

<table>
<tr><th colspan="3">XY_PAT_CHROMA_BLT_IMMEDIATE</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">PAT_BLT_IMMEDIATE is used when there is no source and the color pattern is not trivial (is not a solid color only) and the pattern is pulled through the command stream. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns.
DWL indicates the total number of Dwords of immediate data. All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0
BR00</td><td>31:29</td><td><b>Client</b>
<table>
<tr><td>Default Value:</td><td>02h 2D Processor</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table>
<tr><td>Default Value:</td><td>77h</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>21:20</td><td><b>32bpp Byte Mask</b>
This field is only used for 32bpp.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td><b>[Default]</b></td></tr>
<tr><td>1xb</td><td>Write Alpha Channel</td></tr>
<tr><td>x1b</td><td>Write RGB Channel</td></tr>
</table></td></tr>
<tr><td></td><td>19:17</td><td><b>Transparency Range Mode</b>
(chroma-key) - Dst Chroma-key modes ONLY (SRC ILLEGAL)</td></tr>
<tr><td></td><td>16:15</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>14:12</td><td><b>Pattern Horizontal Seed</b>
Pixel of the scan line to start on corresponding to DST X=0.</td></tr>
<tr><td></td><td>11</td><td><b>Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_PAT_CHROMA_BLT_IMMEDIATE</th></tr>
<tr><td rowspan="2"></td><td>10:8</td><td><b>Pattern Vertical Seed</b>
Scan line of the 8x8 pattern to start on corresponding to DST Y=0.</td></tr>
<tr><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Default Value:</td><td>[22,70] Excludes DWORD 0,1</td></tr>
<tr><td>Format:</td><td>=n</td></tr>
</table>
n = 06 + DWL
Where DWL is immediate data pattern size in dwords.</td></tr>
<tr><td rowspan="6">1
BR13</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>30</td><td><b>Clipping Enabled</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Disabled</td></tr>
<tr><td>1b</td><td>Enabled</td></tr>
</table></td></tr>
<tr><td>29:26</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>25:24</td><td><b>Color Depth</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>8 Bit Color</td></tr>
<tr><td>01b</td><td>16 Bit Color(565)</td></tr>
<tr><td>10b</td><td>16 Bit Color(1555)</td></tr>
<tr><td>11b</td><td>32 Bit Color</td></tr>
</table></td></tr>
<tr><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).</td></tr>
<tr><td rowspan="2">2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
<tr><td rowspan="2">3
BR23</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
16 bit signed number.</td></tr>
<tr><td>15:0</td><td><b>Destination X2 Coordinate (Right)</b>
16 bit signed number.</td></tr>
</table>

XY_PAT_CHROMA_BLT_IMMEDIATE		
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: VIRTUAL_ADDR[63:0]
6 BR18	31:0	**Transparency Color Low** (Chroma-key Low = Pixel Greater or Equal)
7 BR19	31:0	**Transparency Color High** (Chroma-key High = Pixel Less or Equal)
8..n	31:0	**Immediate Data**

XY_PAT_CHROMA_BLT

<table>
<tr><th colspan="3">XY_PAT_CHROMA_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">PAT_BLT is used when there is no source and the color pattern is not trivial (is not a solid color only).
All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0
BR00</td><td>31:29</td><td><b>Client</b>
<table>
<tr><td>Default Value:</td><td>02h 2D Processor</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table>
<tr><td>Default Value:</td><td>76h</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td></td><td>21:20</td><td><b>32bpp Byte Mask</b>
This field is only used for 32bpp.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td><b>[Default]</b></td></tr>
<tr><td>1xb</td><td>Write Alpha Channel</td></tr>
<tr><td>x1b</td><td>Write RGB Channel</td></tr>
</table></td></tr>
<tr><td></td><td>19:17</td><td><b>Transparency Range Mode</b>
(chroma-key) - Dst Chroma-key modes ONLY (SRC ILLEGAL)</td></tr>
<tr><td></td><td>16:15</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>14:12</td><td><b>Pattern Horizontal Seed</b>
Pixel of the scan line to start on corresponding to DST X=0.</td></tr>
<tr><td></td><td>11</td><td><b>Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
<tr><td></td><td>10:8</td><td><b>Pattern Vertical Seed</b>
Scan line of the 8x8 pattern to start on corresponding to DST Y=0.</td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Default Value:</td><td>08h Excludes DWORD 0,1</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_PAT_CHROMA_BLT</th></tr>
<tr><td>1
BR13</td><td>31</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Clipping Enabled</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0b</td><td>Disabled</td></tr><tr><td>1b</td><td>Enabled</td></tr></table></td></tr>
<tr><td></td><td>29:26</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>25:24</td><td><b>Color Depth</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>8 Bit Color</td></tr><tr><td>01b</td><td>16 Bit Color(565)</td></tr><tr><td>10b</td><td>16 Bit Color(1555)</td></tr><tr><td>11b</td><td>32 Bit Color</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td></td><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).</td></tr>
<tr><td>2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
<tr><td>3
BR23</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X2 Coordinate (Right)</b>
16 bit signed number.</td></tr>
<tr><td>4..5</td><td>63:0</td><td><b>Destination Base Address</b><table><tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr></table>This address must be Cache Line (64Byte) aligned for all surface types (linear or tiled).</td></tr>
<tr><td>6..7</td><td>63:0</td><td><b>Pattern Base Address</b><table><tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr></table>(28:06 are implemented) (Note no NPO2 change here). The pattern data must be located in linear memory. The Pattern Base Address must always be Cache Line (64byte) aligned.</td></tr>
<tr><td>8
BR18</td><td>31:0</td><td><b>Transparency Color Low</b>
(Chroma-key Low = Pixel Greater or Equal)</td></tr>
</table>

XY_PAT_CHROMA_BLT		
9 BR19	31:0	**Transparency Color High** (Chroma-key High = Pixel Less or Equal)

XY_PIXEL_BLT

XY_PIXEL_BLT

Source:	BlitterCS
Length Bias:	2

The Destination X coordinate and Destination Y coordinate is compared with the ClipRect registers. If it is within all 4 comparisons, then the pixel supplied in the XY_SETUP_BLT instruction is written with the raster operation to (Destination Y Address + (Destination Y coordinate * Destination pitch) + (Destination X coordinate * bytes per pixel)).
ROP field must specify pattern or fill with 0's or 1's. There is no source operand.
Negative Stride (= Pitch) specified in the Setup command is Not Allowed

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 24h Format: Opcode
	21:12	**Reserved** Access: RO Format: MBZ
	11	**Tiling Enable** Value | Name | Description 0b | Tiling Disabled (Linear Blit) | 1b | Tiling Enabled | : Tile-X or Tile-Y.
	10:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 00h
1 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.

XY_SCANLINES_BLT

XY_SCANLINES_BLT

Source:	BlitterCS
Length Bias:	2

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.
Solid pattern should use the XY_SETUP_MONO_PATTERN_SL_BLT instruction.
ROP field must specify pattern or fill with 0's or 1's. There is no source operand.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 25h Format: Opcode
	21:15	**Reserved** Access: RO Format: MBZ
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** Value | Name | Description 0b | Tiling Disabled (Linear Blit) | 1b | Tiling Enabled | : Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 01h
1 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
2 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.

XY_SETUP_BLT

XY_SETUP_BLT

Source:	BlitterCS
Length Bias:	2

This setup instruction supplies common setup information including clipping coordinates used by the XY commands: XY_PIXEL_BLT, XY_SCANLINE_BLT, XY_TEXT_BLT, and XY_TEXT_BLT_IMMEDIATE.
These are the only instructions that require that state be saved between instructions other than the Clipping parameters. There are 5 dedicated registers to contain the state for the 3 setup BLT instructions (XY_SETUP_BLT, XY_SETUP_MONO_PATTERN_SL_BLT, and XY_SETUP_CLIP_BLT. All other BLTs use a temporary version of these. The 5 double word registers are: DW1 (Setup Control), DW6 (Setup Foreground color), DW5 (Setup Background color), DW7 (Setup Pattern address), and DW4 (Setup Destination Base Address).

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 01h Format: Opcode
	21:20	**32 bpp Byte Mask** **Value** | **Name** 1xb | Write Alpha Channel x1b | Write RGB Channel
	19:12	**Reserved** Access: RO Format: MBZ
	11	**Tiling Enable** **Value** | **Name** 0b | Tiling Disabled (Linear Blit) 1b | Tiling Enabled (Tile-X or Tile-Y)
	10:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 08h
1 BR01	31	**Reserved** Access: RO Format: MBZ

<table>
<tr><th colspan="3">XY_SETUP_BLT</th></tr>
<tr><td rowspan="6"></td><td>30</td><td><b>Clipping Enabled</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0b</td><td>Disabled</td></tr><tr><td>1b</td><td>Enabled</td></tr></table></td></tr>
<tr><td>29</td><td><b>Mono Source Transparency Mode</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0b</td><td>Use Background</td></tr><tr><td>1b</td><td>Transparency Enabled</td></tr></table></td></tr>
<tr><td>28:26</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>25:24</td><td><b>Color Depth</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>8 Bit Color</td></tr><tr><td>01b</td><td>16 Bit Color(565)</td></tr><tr><td>10b</td><td>16 Bit Color(1555)</td></tr><tr><td>11b</td><td>32 Bit Color</td></tr></table></td></tr>
<tr><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).</td></tr>
<tr><td rowspan="2">2
BR24</td><td>31:16</td><td><b>ClipRect Y1 Coordinate (Top)</b>
(30:16 = 15 bit positive number)</td></tr>
<tr><td>15:0</td><td><b>ClipRect X1 Coordinate (Left)</b>
(14:00 = 15 bit positive number)</td></tr>
<tr><td rowspan="2">3
BR25</td><td>31:16</td><td><b>ClipRect Y2 Coordinate (Bottom)</b>
(30:16 = 15 bit positive number)</td></tr>
<tr><td>15:0</td><td><b>ClipRect X2 Coordinate (Right)</b>
(14:00 = 15 bit positive number)</td></tr>
<tr><td>4..5
Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.</td><td>63:0</td><td><b>Destination Base Address</b><table><tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr></table></td></tr>
<tr><td>6
BR05</td><td>31:0</td><td><b>Setup Background Color</b>
8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] All</td></tr>
</table>

<table>
<tr><th colspan="3">XY_SETUP_BLT</th></tr>
<tr><td>7
BR06</td><td>31:0</td><td><b>Setup Foreground Color</b>
8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] (SLB and TB only)</td></tr>
<tr><td>8..9
28:06 are implemented. Note no NPO2 change here. The pattern data must be located in linear memory. The Setup Pattern Base Address for Color Pattern must always be Cache Line (64byte) aligned.</td><td>63:0</td><td><b>Setup Pattern Base Address for Color Pattern</b><table><tr><td>Format:</td><td>VIRTUAL_ADDR[63:0]</td></tr></table></td></tr>
</table>

XY_SETUP_CLIP_BLT

XY_SETUP_CLIP_BLT

Source:	BlitterCS
Length Bias:	2

This command is used to only change the clip coordinate registers. These are the same clipping registers as the Setup clipping registers above.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 03h Format: Opcode
	21:12	**Reserved** Access: RO Format: MBZ
	11	**Tiling Enable** **Value** / **Name** 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled (Tile-X or Tile-Y
	10:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 01h
1 BR24	31:16	**ClipRect Y1 Coordinate (Top)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X1 Coordinate (Left)** (14:00 = 15 bit positive number)
2 BR25	31:16	**ClipRect Y2 Coordinate (Bottom)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X2 Coordinate (Right)** (14:00 = 15 bit positive number)

XY_SETUP_MONO_PATTERN_SL_BLT

XY_SETUP_MONO_PATTERN_SL_BLT

Source:	BlitterCS
Length Bias:	2

This setup instruction supplies common setup information including clipping coordinates used exclusively with the following instruction: XY_SCANLINE_BLT (SLB) - 1 scan line of monochrome pattern and destination are the only operands allowed.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 11h Format: Opcode
	21:20	**32 bpp Byte Mask** Value | Name 1xb | Write Alpha Channel x1b | Write RGB Channel
	19:12	**Reserved** Access: RO Format: MBZ
	11	**Tiling Enable** Value | Name 0b | Tiling Disabled (Linear Blit) 1b | Tiling Enabled (Tile-X or Tile-Y
	10:8	**Reserved** Access: RO Format: MBZ
	7:0	**DWord Length** Default Value: 08h
1 BR01	31	**Solid Pattern Select** (SLB and Pixel only) Value | Name 0 | No Solid Pattern 1 | Solid Pattern

<table>
<tr><th colspan="3">XY_SETUP_MONO_PATTERN_SL_BLT</th></tr>
<tr><td rowspan="8"></td><td>30</td><td><b>Clipping Enabled</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Disabled</td></tr>
<tr><td>1b</td><td>Enabled</td></tr>
</table></td></tr>
<tr><td>29</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>28</td><td><b>Mono Pattern Transparency Mode</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Use Background</td></tr>
<tr><td>1b</td><td>Transparency Enabled</td></tr>
</table></td></tr>
<tr><td>27:26</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>25:24</td><td><b>Color Depth</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>8 Bit Color</td></tr>
<tr><td>01b</td><td>16 Bit Color(565)</td></tr>
<tr><td>10b</td><td>16 Bit Color(1555)</td></tr>
<tr><td>11b</td><td>32 Bit Color</td></tr>
</table></td></tr>
<tr><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be up to 128Kbytes (or 32KDwords).</td></tr>
<tr></tr>
<tr><td rowspan="2">2
BR24</td><td>31:16</td><td><b>ClipRect Y1 Coordinate (Top)</b>
(30:16 = 15 bit positive number)</td></tr>
<tr><td>15:0</td><td><b>ClipRect X1 Coordinate (Left)</b>
(14:00 = 15 bit positive number)</td></tr>
<tr><td rowspan="2">3
BR25</td><td>31:16</td><td><b>ClipRect Y2 Coordinate (Bottom)</b>
(30:16 = 15 bit positive number)</td></tr>
<tr><td>15:0</td><td><b>ClipRect X2 Coordinate (Right)</b>
(14:00 = 15 bit positive number)</td></tr>
</table>

XY_SETUP_MONO_PATTERN_SL_BLT

4..5 This bitfield contains the base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Setup Destination Base Address** Format: VIRTUAL_ADDR[63:0]
6 BR05	31:0	**Setup Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] All
7 BR06	31:0	**Setup Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] (SLB and TB only)
8 BR20	31:0	**DW0 (least significant) for a Monochrome Pattern**
9 BR21	31:0	**DW1 (most significant) for a Monochrome Pattern**

XY_SRC_COPY_BLT

<table>
<tr><th colspan="3">XY_SRC_COPY_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">This BLT instruction performs a color source copy where the only operands involved is a color source and destination of the same bit width.
The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access. The ROP value chosen must involve source and no pattern data in the ROP operation.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="6">0
BR00</td><td>31:29</td><td><b>Client</b>
<table>
<tr><td>Default Value:</td><td>02h 2D Processor</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table>
<tr><td>Default Value:</td><td>53h</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td>21:20</td><td><b>32bpp Byte Mask</b>
This field is only used for 32bpp.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td><b>[Default]</b></td></tr>
<tr><td>1xb</td><td>Write Alpha Channel</td></tr>
<tr><td>x1b</td><td>Write RGB Channel</td></tr>
</table></td></tr>
<tr><td>19:16</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>15</td><td><b>Src Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
<tr><td>14:12</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_SRC_COPY_BLT</th></tr>
<tr><td></td><td>11</td><td><b>Dest Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
<tr><td></td><td>10:8</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b>
<table>
<tr><td>Format:</td><td>=n</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>08h</td><td></td></tr>
</table></td></tr>
<tr><td>1
BR13</td><td>31</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>30</td><td><b>Clipping Enabled</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0b</td><td>Disabled</td></tr>
<tr><td>1b</td><td>Enabled</td></tr>
</table></td></tr>
<tr><td></td><td>29:26</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>25:24</td><td><b>Color Depth</b>
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td>8 Bit Color</td></tr>
<tr><td>01b</td><td>16 Bit Color(565)</td></tr>
<tr><td>10b</td><td>16 Bit Color(1555)</td></tr>
<tr><td>11b</td><td>32 Bit Color</td></tr>
</table></td></tr>
<tr><td></td><td>23:16</td><td><b>Raster Operation</b>
It identifies the bit-wise operations that needs to be performed. Details of bit-wise operations can be found in Bit-Wise Operations.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be up to 128Kbytes (or 32KDwords).</td></tr>
<tr><td>2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
</table>

XY_SRC_COPY_BLT

	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: VIRTUAL_ADDR[63:0]
6 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
7 BR11	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Source Pitch (double word aligned) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Yand can be up to 128Kbytes (or 32KDwords).
8..9 Base address of the source surface: X=0, Y=0. When Src Tiling is enabled (Bit_15 enabled), this address must be 4KB-aligned. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Source Base Address** Format: VIRTUAL_ADDR[63:0]

XY_SRC_COPY_CHROMA_BLT

<table>
<tr><th colspan="3">XY_SRC_COPY_CHROMA_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">This BLT instruction performs a color source copy with chroma-keying where the only operands involved is a color source and destination of the same bit width.
The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.
The ROP value chosen must involve source and no pattern data in the ROP operation.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="7">0
BR00</td><td>31:29</td><td><b>Client</b>
<table>
<tr><td>Default Value:</td><td>02h 2D Processor</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table>
<tr><td>Default Value:</td><td>73h</td></tr>
<tr><td>Format:</td><td>Opcode</td></tr>
</table></td></tr>
<tr><td>21:20</td><td><b>32bpp Byte Mask</b>
This field is only used for 32bpp.
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>00b</td><td><b>[Default]</b></td></tr>
<tr><td>1xb</td><td>Write Alpha Channel</td></tr>
<tr><td>x1b</td><td>Write RGB Channel</td></tr>
</table></td></tr>
<tr><td>19:17</td><td><b>Transparency Range Mode</b>
(chroma-key)</td></tr>
<tr><td>16</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td>15</td><td><b>Src Tiling Enable</b>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0b</td><td>Tiling Disabled (Linear)</td><td></td></tr>
<tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr>
</table></td></tr>
<tr><td>14:12</td><td><b>Reserved</b>
<table>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="3">XY_SRC_COPY_CHROMA_BLT</th></tr>
<tr><td></td><td>11</td><td><b>Dest Tiling Enable</b><table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr><tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr></table></td></tr>
<tr><td></td><td>10:8</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>7:0</td><td><b>DWord Length</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0Ah</td><td></td></tr></table></td></tr>
<tr><td>1
BR13</td><td>31</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>30</td><td><b>Clipping Enabled</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>0b</td><td>Disabled</td></tr><tr><td>1b</td><td>Enabled</td></tr></table></td></tr>
<tr><td></td><td>29:26</td><td><b>Reserved</b><table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>25:24</td><td><b>Color Depth</b><table><tr><th>Value</th><th>Name</th></tr><tr><td>00b</td><td>8 Bit Color</td></tr><tr><td>01b</td><td>16 Bit Color(565)</td></tr><tr><td>10b</td><td>16 Bit Color(1555)</td></tr><tr><td>11b</td><td>32 Bit Color</td></tr></table></td></tr>
<tr><td></td><td>23:16</td><td><b>Raster Operation</b></td></tr>
<tr><td></td><td>15:0</td><td><b>Destination Pitch in DWords</b>
2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).</td></tr>
<tr><td>2
BR22</td><td>31:16</td><td><b>Destination Y1 Coordinate (Top)</b>
16 bit signed number.</td></tr>
<tr><td></td><td>15:0</td><td><b>Destination X1 Coordinate (Left)</b>
16 bit signed number.</td></tr>
<tr><td>3
BR23</td><td>31:16</td><td><b>Destination Y2 Coordinate (Bottom)</b>
16 bit signed number.</td></tr>
</table>

XY_SRC_COPY_CHROMA_BLT		
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4..5 Base address of the destination surface: X=0, Y=0. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Destination Base Address** Format: VIRTUAL_ADDR[63:0]
6 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
7 BR11	31:16	**Reserved** Access: RO Format: MBZ
	15:0	**Source Pitch (double word aligned) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be up to 128Kbytes (or 32KDwords).
8..9 Base address of the source surface: X=0, Y=0. When Src Tiling is enabled (Bit_15 enabled), this address must be 4KB-aligned. When Tiling is not enabled, this address is to be CL (64byte) aligned.	63:0	**Source Base Address** Format: VIRTUAL_ADDR[63:0]
10 BR18	31:0	**Transparency Color Low** (Chroma-key Low = Pixel Greater or Equal)
11 BR19	31:0	**Transparency Color High** (Chroma-key High = Pixel Less or Equal)

XY_TEXT_BLT

XY_TEXT_BLT

Source:	BlitterCS
Length Bias:	2

All source scan lines and pixels that fall within the ClipRect Y and X coordinates are written. The source address corresponds to Destination X1 and Y1 coordinate.
Text is either bit or byte packed. Bit packed means that the next scan line starts 1 pixel after the end of the current scan line with no bit padding. Byte packed means that the next scan line starts on the first bit of the next byte boundary after the last bit of the current line.
Source expansion color registers are always in the SETUP_BLT.
Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 26h Format: Opcode
	21:17	**Reserved** Access: RO Format: MBZ
	16	**Bit / Byte Packed** Byte packed is for the NT driver. Value / Name: 0 Bit 1 Byte
	15:12	**Reserved** Access: RO Format: MBZ
	11	**Tiling Enable** Value / Name / Description: 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled : Tile-X or Tile-Y.
	10:8	**Reserved** Access: RO Format: MBZ

XY_TEXT_BLT		
	7:0	**DWord Length** Default Value: 03h
1 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
2 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
3..4 Address of the first byte on a scan line corresponding to source X1, Y1. Note no NPO2 change here. The source address must always be Cache Line (64byte) aligned.	63:0	**Source Address** Format: VIRTUAL_ADDR[63:0]

XY_TEXT_IMMEDIATE_BLT

<table>
<tr><th colspan="3">XY_TEXT_IMMEDIATE_BLT</th></tr>
<tr><td colspan="3">Source: BlitterCS
Length Bias: 2</td></tr>
<tr><td colspan="3">This instruction allows the Driver to send data through the instruction stream that eliminates the read latency of reading a source from memory.
If an operand is in system cacheable memory and either small or only accessed once, it can be copied directly to the instruction stream versus to graphics accessible memory. The IMMEDIATE_BLT data MUST transfer an even number of doublewords.
The BLT engine will hang if it does not get an even number of doublewords. All source scan lines and pixels that fall within the ClipRect X and Y coordinates are written. The source data corresponds to Destination X1 and Y1 coordinate.
Source expansion color registers are always in the SETUP_BLT. NEGATIVE STRIDE (= PITCH) IS NOT ALLOWED.</td></tr>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0
BR00</td><td>31:29</td><td><b>Client</b>
<table><tr><td>Default Value:</td><td>02h 2D Processor</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>28:22</td><td><b>Instruction Target(Opcode)</b>
<table><tr><td>Default Value:</td><td>31h</td></tr><tr><td>Format:</td><td>Opcode</td></tr></table></td></tr>
<tr><td></td><td>21:17</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>16</td><td><b>Bit / Byte Packed</b>
Byte packed is for the NT driver.
<table><tr><th>Value</th><th>Name</th></tr><tr><td>0</td><td>Bit</td></tr><tr><td>1</td><td>Byte</td></tr></table></td></tr>
<tr><td></td><td>15:12</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>11</td><td><b>Tiling Enable</b>
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0b</td><td>Tiling Disabled (Linear Blit)</td><td></td></tr><tr><td>1b</td><td>Tiling Enabled</td><td>: Tile-X or Tile-Y.</td></tr></table></td></tr>
<tr><td></td><td>10:8</td><td><b>Reserved</b>
<table><tr><td>Access:</td><td>RO</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

XY_TEXT_IMMEDIATE_BLT

	7:0	**DWord Length** Default Value: | [17,65] Excludes DWORD 0,1 Format: | =n n = 01 + DWL Where DWL indicates size of indirect data in dwords.
1 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
2 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
3..n	31:0	**Immediate Data**