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"

Volume 2b: Command Reference: Enumerations

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

3D_Color_Buffer_Blend_Factor

3D_Color_Buffer_Blend_Factor	
Size (in bits):	5

Value	Name
00h	Reserved
01h	BLENDFACTOR_ONE
02h	BLENDFACTOR_SRC_COLOR
03h	BLENDFACTOR_SRC_ALPHA
04h	BLENDFACTOR_DST_ALPHA
05h	BLENDFACTOR_DST_COLOR
06h	BLENDFACTOR_SRC_ALPHA_SATURATE
07h	BLENDFACTOR_CONST_COLOR
08h	BLENDFACTOR_CONST_ALPHA
09h	BLENDFACTOR_SRC1_COLOR
0Ah	BLENDFACTOR_SRC1_ALPHA
0Bh-10h	Reserved
11h	BLENDFACTOR_ZERO
12h	BLENDFACTOR_INV_SRC_COLOR
13h	BLENDFACTOR_INV_SRC_ALPHA
14h	BLENDFACTOR_INV_DST_ALPHA
15h	BLENDFACTOR_INV_DST_COLOR
16h	Reserved
17h	BLENDFACTOR_INV_CONST_COLOR
18h	BLENDFACTOR_INV_CONST_ALPHA
19h	BLENDFACTOR_INV_SRC1_COLOR
1Ah	BLENDFACTOR_INV_SRC1_ALPHA

3D_Color_Buffer_Blend_Function

3D_Color_Buffer_Blend_Function		
Size (in bits): 3		
Value	**Name**	**Description**
0	BLENDFUNCTION_ADD	BLENDFUNCTION_ADD
1	BLENDFUNCTION_SUBTRACT	BLENDFUNCTION_SUBTRACT
2	BLENDFUNCTION_REVERSE_SUBTRACT	BLENDFUNCTION_REVERSE_SUBTRACT
3	BLENDFUNCTION_MIN	BLENDFUNCTION_MIN
4	BLENDFUNCTION_MAX	BLENDFUNCTION_MAX
5 - 7	Reserved	

3D_Compare_Function

3D_Compare_Function		
Size (in bits): 3		
Value	**Name**	**Description**
0h	COMPAREFUNCTION_ALWAYS	Always pass
1h	COMPAREFUNCTION_NEVER	Never pass
2h	COMPAREFUNCTION_LESS	Pass if the value is less than the reference
3h	COMPAREFUNCTION_EQUAL	Pass if the value is equal to the reference
4h	COMPAREFUNCTION_LEQUAL	Pass if the value is less than or equal to the reference
5h	COMPAREFUNCTION_GREATER	Pass if the value is greater than the reference
6h	COMPAREFUNCTION_NOTEQUAL	Pass if the value is not equal to the reference
7h	COMPAREFUNCTION_GEQUAL	Pass if the value is greater than or equal to the reference

3D_Logic_Op_Function

3D_Logic_Op_Function		
Size (in bits): 4		
Value	**Name**	**Description**
0h	LOGICOP_CLEAR	BLACK; all 0's
1h	LOGICOP_NOR	NOTMERGEPEN; NOT (S OR D)
2h	LOGICOP_AND_INVERTED	MASKNOTPEN; (NOT S) AND D
3h	LOGICOP_COPY_INVERTED	NOTCOPYPEN; NOT S
4h	LOGICOP_AND_REVERSE	MASKPENNOT; S AND NOT D
5h	LOGICOP_INVERT	NOT; NOT D
6h	LOGICOP_XOR	XORPEN; S XOR D
7h	LOGICOP_NAND	NOTMASKPEN; NOT (S AND D)
8h	LOGICOP_AND	MASKPEN; S AND D
9h	LOGICOP_EQUIV	NOTXORPEN; NOT (S XOR D)
Ah	LOGICOP_NOOP	NOP; D
Bh	LOGICOP_OR_INVERTED	MERGENOTPEN; (NOT S) OR D
Ch	LOGICOP_COPY	COPYPEN; S
Dh	LOGICOP_OR_REVERSE	MERGEPENNOT; S OR NOT D
Eh	LOGICOP_OR	MERGEPEN; S OR D
Fh	LOGICOP_SET	WHITE; all 1's

3D_Prim_Topo_Type

3D_Prim_Topo_Type	
Source:	RenderCS
Size (in bits):	6

The following table defines the encoding of the Primitive Topology Type field. See 3D Pipeline for details, programming restrictions, diagrams and a discussion of the basic primitive types.

Value	Name	Description
00h	Reserved	
01h	3DPRIM_POINTLIST	
02h	3DPRIM_LINELIST	
03h	3DPRIM_LINESTRIP	
04h	3DPRIM_TRILIST	
05h	3DPRIM_TRISTRIP	
06h	3DPRIM_TRIFAN	
07h	3DPRIM_QUADLIST	
08h	3DPRIM_QUADSTRIP	
09h	3DPRIM_LINELIST_ADJ	
0Ah	3DPRIM_LINESTRIP_ADJ	
0Bh	3DPRIM_TRILIST_ADJ	
0Ch	3DPRIM_TRISTRIP_ADJ	
0Dh	3DPRIM_TRISTRIP_REVERSE	
0Eh	3DPRIM_POLYGON	
0Fh	3DPRIM_RECTLIST	
10h	3DPRIM_LINELOOP	
11h	Reserved	Reserved for HW use as POINTLIST_BF
12h	3DPRIM_LINESTRIP_CONT	
13h	Reserved	Reserved for HW use as LINESTRIP_BF
14h	Reserved	Reserved for HW use as LINESTRIP_CONT_BF
15h	Reserved	
16h	3DPRIM_TRIFAN_NOSTIPPLE	
17h	Reserved	
18h	Reserved	
19h	Reserved	
1Ah	Reserved	
[1Bh-1Fh]	Reserved	
20h	3DPRIM_PATCHLIST_1	List of 1-vertex patches

3D_Prim_Topo_Type		
21h	3DPRIM_PATCHLIST_2	
22h	3DPRIM_PATCHLIST_3	
23h	3DPRIM_PATCHLIST_4	
24h	3DPRIM_PATCHLIST_5	
25h	3DPRIM_PATCHLIST_6	
26h	3DPRIM_PATCHLIST_7	
27h	3DPRIM_PATCHLIST_8	
28h	3DPRIM_PATCHLIST_9	
29h	3DPRIM_PATCHLIST_10	
2ah	3DPRIM_PATCHLIST_11	
2bh	3DPRIM_PATCHLIST_12	
2ch	3DPRIM_PATCHLIST_13	
2dh	3DPRIM_PATCHLIST_14	
2eh	3DPRIM_PATCHLIST_15	
2fh	3DPRIM_PATCHLIST_16	
30h	3DPRIM_PATCHLIST_17	
31h	3DPRIM_PATCHLIST_18	
32h	3DPRIM_PATCHLIST_19	
33h	3DPRIM_PATCHLIST_20	
34h	3DPRIM_PATCHLIST_21	
35h	3DPRIM_PATCHLIST_22	
36h	3DPRIM_PATCHLIST_23	
37h	3DPRIM_PATCHLIST_24	
38h	3DPRIM_PATCHLIST_25	
39h	3DPRIM_PATCHLIST_26	
3ah	3DPRIM_PATCHLIST_27	
3bh	3DPRIM_PATCHLIST_28	
3ch	3DPRIM_PATCHLIST_29	
3dh	3DPRIM_PATCHLIST_30	
3eh	3DPRIM_PATCHLIST_31	
3Fh	3DPRIM_PATCHLIST_32	List of 32-vertex patches

3D_Stencil_Operation

3D_Stencil_Operation	
Source:	RenderCS
Size (in bits):	3

Value	Name
0	STENCILOP_KEEP
1	STENCILOP_ZERO
2	STENCILOP_REPLACE
3	STENCILOP_INCRSAT
4	STENCILOP_DECRSAT
5	STENCILOP_INCR
6	STENCILOP_DECR
7	STENCILOP_INVERT

3D_Vertex_Component_Control

3D_Vertex_Component_Control	
Source:	RenderCS
Size (in bits):	3

Value	Name	Description
0	VFCOMP_NOSTORE	Don't store this component. (Not valid for Component 0, but can be used for Component 1-3). Once this setting is used for a component, all higher-numbered components (if any) MUST also use this setting. (I.e., no holes within any particular vertex element). VFCOMP_NOSTORE will not store a component if the SourceElementFormat is R64_PASSTHRU or R64G64_PASSTHRU and it is used on component 2 and 3 else 0 will be stored.
1	VFCOMP_STORE_SRC	Store corresponding component from format-converted source element. Storing a component that is not included in the Source Element Format results in an UNPREDICTABLE value being stored. VF will process Component Control fields within a VERTEX_ELEMENT_STATE structure sequentially, starting with Component 0 Control. For each Component Control field in this sequence, when VF detects (a) the Component Control field is set to STORE_SRC and (b) the component is not overwritten by an SGV, VF will store a component of the source vertex data into the destination component. The first such Component Control field satisfying this criterion will use Component 0 of the source vertex data, the second such Component Control field will use Component 1 of the source vertex data, and so on. Therefore, when a lower-numbered Component Control field (a) is set to something other than STORE_SRC (e.g., STORE_0) or (b) the component is overwritten with an SGV, the source vertex component used when a higher-numbered Component Control fields is set to STORE_SRC will be impacted.
2	VFCOMP_STORE_0	Store 0 (interpreted as 0.0f if accessed as a float value)
3	VFCOMP_STORE_1_FP	Store 1.0f
4	VFCOMP_STORE_1_INT	Store 0x1
5-6	-	Reserved
7	VFCOMP_STORE_PID	Store Primitive ID (as U32) Software can no longer use this encoding as PrimitiveID is passed down the FF pipeline - see explanation above.

AccWrCtrl

AccWrCtrl	
Size (in bits):	1
This field allows per instruction accumulator write control. When set, the result is written to both destination and accumulator.	

Value	Name
0	Don't write to ACC **[Default]**
1	Update ACC

AddrMode

AddrMode	
Source:	EuIsa
Size (in bits):	1

Addressing Mode This field determines the addressing method of the operand. Normally the destination operand and each source operand each have a distinct addressing mode field. When it is cleared, the register address of the operand is directly provided by bits in the instruction word. It is called a direct register addressing mode. When it is set, the register address of the operand is computed based on the address register value and an address immediate field in the instruction word. This is referred to as a register-indirect register addressing mode. This field applies to the destination operand and the first source operand, src0. Support for src1 is device dependent. See Table XX (Indirect source addressing support available in device hardware) in ISA Execution Environment for details.

Programming Notes
Instructions with 3 source operands use Direct Addressing.

Value	Name	Description
0	Direct	'Direct' register addressing
1	Indirect	'Register-Indirect' (or in short 'Indirect'). Register-indirect register addressing

AtomicCtrl

AtomicCtrl	
Source:	EuIsa
Size (in bits):	1

Enables the atomic instruction control option

Value	Name	Description
0b	No Operation **[Default]**	This value leaves thread switching up to the EU's scheduler.
1b	Atomic	Prevent any thread switch immediately following this instruction. Always execute the next instruction (which may not be next sequentially if the current instruction branches). The next instruction gets highest priority in the thread arbitration for the execution pipelines.

Attribute_Component_Format

Attribute_Component_Format	
Source:	RenderCS
Size (in bits):	2

Value	Name	Description
00b	disabled **[Default]**	All components disabled
01b	.xy	2D attribute, z and w components disabled
10b	.xyz	3D attribute, w components disabled
11b	.xyzw	4D attribute, no disabled components

BARRIER_SIZE

BARRIER_SIZE - BARRIER_SIZE	
Size (in bits):	1
Specifies number of barriers in the threadgroup.	

Value	Name
0	None **[Default]**
1	B1

BooleanFuncCtrl

BooleanFuncCtrl			
Size (in bits): 8			
This enumeration describes all the value lookup indices for the bfn instruction. See bfn for how this value is determined. The symbols s0, s1, and s2 correspond to the operands src0, src1, and src2 in the descriptions below.			
Value	**Name**	**Description**	
00000000b	Boolean Function 0x00	This symbol will compute `0` (the zero function)	Syntax: TERNARY_REGIMM_REG_REGIMM
00000001b	Boolean Function 0x01	This symbol will compute `~s0&~s1&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00000010b	Boolean Function 0x02	This symbol will compute `s0&~s1&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00000011b	Boolean Function 0x03	This symbol will compute `~s1&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00000100b	Boolean Function 0x04	This symbol will compute `~s0&s1&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00000101b	Boolean Function 0x05	This symbol will compute `~s0&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00000110b	Boolean Function 0x06	This symbol will compute `(s0^s1)&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00000111b	Boolean Function 0x07	This symbol will compute `(~s0|~s1)&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00001000b	Boolean Function 0x08	This symbol will compute `s0&s1&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00001001b	Boolean Function 0x09	This symbol will compute `(s0^~s1)&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00001010b	Boolean Function 0x0A	This symbol will compute `s0&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00001011b	Boolean Function 0x0B	This symbol will compute `(s0|~s1)&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00001100b	Boolean Function 0x0C	This symbol will compute `s1&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00001101b	Boolean Function 0x0D	This symbol will compute `(~s0|s1)&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00001110b	Boolean Function 0x0E	This symbol will compute `(s0|s1)&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl			
00001111b	Boolean Function 0x0F	This symbol will compute `~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00010000b	Boolean Function 0x10	This symbol will compute `~s0&~s1&s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00010001b	Boolean Function 0x11	This symbol will compute `~s0&~s1`	Syntax: TERNARY_REGIMM_REG_REGIMM
00010010b	Boolean Function 0x12	This symbol will compute `(s0^s2)&~s1`	Syntax: TERNARY_REGIMM_REG_REGIMM
00010011b	Boolean Function 0x13	This symbol will compute `(~s0|~s2)&~s1`	Syntax: TERNARY_REGIMM_REG_REGIMM
00010100b	Boolean Function 0x14	This symbol will compute `~s0&(s1^s2)`	Syntax: TERNARY_REGIMM_REG_REGIMM
00010101b	Boolean Function 0x15	This symbol will compute `~s0&(~s1|~s2)`	Syntax: TERNARY_REGIMM_REG_REGIMM

00010110b	Boolean Function 0x16	This symbol will compute the following functions:
		`s0^(s0&s1|s1^s2)`
		`s0^(s0&s2|s1^s2)`
		`(s0|s1)^(s0&s1|s2)`
		`(s0|s2)^(s0&s2|s1)`
		`(s0|s1&s2)^(s1|s2)`
		`(s0^s1|s0&s2)^s2`
		`(s0^s1|s1&s2)^s2`
		`(s0^s2|s0&s1)^s1`
		`(s0^s2|s1&s2)^s1`
		`s0^(s0&s1|~s2)^~s1`
		`s0^(s0&s2|~s1)^~s2`
		`s0^(~s0|~s1)&s2^s1`
		`s0^(~s0|~s2)&s1^s2`
		`(~s0|s1&s2)^s1^~s2`
		`(~s0|s1&s2)^~s1^s2`
		`s0&(~s1|~s2)^s1^s2`
		`s0&(~s1|~s2)^~s1^~s2`
		`~s0&~s1^(~s0|~s1)&~s2`
		`~s0&~s2^(~s0|~s2)&~s1`
		`~s0&(~s1|~s2)^~s1&~s2`
		`(s0^~s1)&(~s0|~s2)^~s2`
		`(s0^~s1)&(~s1|~s2)^~s2`
		`(s0^~s2)&(~s0|~s1)^~s1`
		`(s0^~s2)&(~s1|~s2)^~s1`
		`(~s0|~s1)&((s0|s1)^s2)`
		`(~s0|~s1)&(s0^s1^s2)`
		`(~s0|~s1)&(s0^~s1^~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		(~s0|~s1)&(~s0&~s1^~s2) (~s0|~s2)&((s0|s2)^s1) (~s0|~s2)&(s0^s1^s2) (~s0|~s2)&(s0^~s1^~s2) (~s0|~s2)&(~s0&~s2^~s1) (s0^(s1|s2))&(~s1|~s2) (s0^s1^s2)&(~s1|~s2) (s0^~s1^~s2)&(~s1|~s2)	
00010111b	Boolean Function 0x17	This symbol will compute the following functions: ~s0&~s1|(~s0|~s1)&~s2 ~s0&~s1|(s0^s1)&~s2 ~s0&~s2|(~s0|~s2)&~s1 ~s0&~s2|(s0^s2)&~s1 ~s0&(~s1|~s2)|~s1&~s2 ~s0&(s1^s2)|~s1&~s2 s0^(s0^~s1|s1^s2) s0^(s0^~s2|s1^s2) (s0|s1)^(s0^~s1|s2) (s0|s2)^(s0^~s2|s1) (s0|s1^~s2)^(s1|s2) (~s0|~s1)^(s0^s1)&s2 (~s0|~s2)^(s0^s2)&s1 (~s0|s1^~s2)^s1&s2 (s0^s1|s1^~s2)^s2 (s0^s2|s1^~s2)^s1 (s0^~s1|s1^~s2)^s1 (s0^~s2|s1^~s2)^s2 s0&s1^(s0^~s1|~s2) s0&s2^(s0^~s2|~s1) s0&(s1^s2)^(~s1|~s2) ~s0&~s1^(s0^s1)&~s2 ~s0&~s2^(s0^s2)&~s1 ~s0&(s1^s2)^~s1&~s2 (s0^s1)&(s1^s2)^~s1 (s0^s2)&(s1^s2)^~s2 (s0^~s1)&(s1^s2)^~s2 (s0^~s2)&(s1^s2)^~s1 (~s0|~s1)&(s0^~s1|~s2) (~s0|~s1)&(~s0&~s1|~s2) (~s0|~s2)&(s0^~s2|~s1) (~s0|~s2)&(~s0&~s2|~s1) (~s0|s1^~s2)&(~s1|~s2) (~s0|~s1&~s2)&(~s1|~s2)	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl			

00011000b	Boolean Function 0x18	This symbol will compute the following functions:
		`(s0^s2)&(s1^s2)`
		`(s0^~s1)&(s1^s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

00011001b	Boolean Function 0x19	This symbol will compute the following functions:
		`s0^(s0&s2|~s1)`
		`(~s0|s1&s2)^s1`
		`s0&(~s1|~s2)^~s1`
		`(s0^~s1)&(~s0|~s2)`
		`(s0^~s1)&(~s1|~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

00011010b	Boolean Function 0x1A	This symbol will compute the following functions:
		`s0^(s0|~s1)&s2`
		`(s0|s1&s2)^s2`
		`~s0&(~s1|~s2)^~s2`
		`(s0|~s1)&(s0^s2)`
		`(s0^s2)&(~s1|~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

00011011b	Boolean Function 0x1B	This symbol will compute the following functions:
		`s0&~s2|~s0&~s1`
		`(s0|s1)^(~s0|s2)`
		`(s0|~s1)^s0&s2`
		`(s0|s1^~s2)^s2`
		`(~s0|~s2)^~s0&s1`
		`(~s0|s1^~s2)^s1`
		`s0&~s2^~s0&~s1`
		`s0&(s1^s2)^~s1`
		`~s0&(s1^s2)^~s2`
		`(s0|~s1)&(~s0|~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

00011100b	Boolean Function 0x1C	This symbol will compute the following functions:
		`(s0&s2|s1)^s2`
		`(s0&~s1|~s2)^~s1`
		`(~s0|s1)&s2^s1`
		`(~s0|~s2)&~s1^~s2`
		`(~s0|s1)&(s1^s2)`
		`(~s0|~s2)&(s1^s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl			

00011101b	Boolean Function 0x1D	This symbol will compute the following functions:
		`~s0&~s1|s1&~s2`
		`s0^(s0^~s2|~s1)`
		`(s0|s1)^(~s1|s2)`
		`(~s0|s1)^s1&s2`
		`(s0^~s2|s1)^s2`
		`s0&~s1^(~s1|~s2)`
		`~s0&~s1^s1&~s2`
		`(s0^s2)&~s1^~s2`
		`(~s0|s1)&(~s1|~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

00011110b	Boolean Function 0x1E	This symbol will compute the following functions:
		`(s0|s1)^s2`
		`~s0&~s1^~s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

00011111b	Boolean Function 0x1F	This symbol will compute `~s0&~s1|~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00100000b	Boolean Function 0x20	This symbol will compute `s0&~s1&s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
00100001b	Boolean Function 0x21	This symbol will compute `(s0^~s2)&~s1`	Syntax: TERNARY_REGIMM_REG_REGIMM
00100010b	Boolean Function 0x22	This symbol will compute `s0&~s1`	Syntax: TERNARY_REGIMM_REG_REGIMM
00100011b	Boolean Function 0x23	This symbol will compute `(s0|~s2)&~s1`	Syntax: TERNARY_REGIMM_REG_REGIMM

00100100b	Boolean Function 0x24	This symbol will compute the following functions:
		`(s0^s1)&(s1^s2)`
		`(s0^~s2)&(s1^s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

00100101b	Boolean Function 0x25	This symbol will compute the following functions:
		`s0^(s0&s1|~s2)`
		`(~s0|s1&s2)^s2`
		`s0&(~s1|~s2)^~s2`
		`(s0^~s2)&(~s0|~s1)`
		`(s0^~s2)&(~s1|~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl			

00100110b	Boolean Function 0x26	This symbol will compute the following functions:
		`s0^(s0|~s2)&s1`
		`(s0|s1&s2)^s1`
		`~s0&(~s1|~s2)^~s1`
		`(s0|~s2)&(s0^s1)`
		`(s0^s1)&(~s1|~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

00100111b	Boolean Function 0x27	This symbol will compute the following functions:
		`s0&~s1|~s0&~s2`
		`(s0|s2)^(~s0|s1)`
		`(s0|~s2)^s0&s1`
		`(s0|s1^~s2)^s1`
		`(~s0|~s1)^~s0&s2`
		`(~s0|s1^~s2)^s2`
		`s0&~s1^~s0&~s2`
		`s0&(s1^s2)^~s2`
		`~s0&(s1^s2)^~s1`
		`(s0|~s2)&(~s0|~s1)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

00101000b	Boolean Function 0x28	This symbol will compute `s0&(s1^s2)`	Syntax: TERNARY_REGIMM_REG_REGIMM

00101001b	Boolean Function 0x29	This symbol will compute the following functions:
		`s0^(s0|~s1)&(s1^~s2)`
		`s0^(s0|~s2)&(s1^~s2)`
		`(~s0|s1)^(~s0&s1|s2)`
		`(~s0|s2)^(~s0&s2|s1)`
		`(~s0|s1&s2)^(s1|s2)`
		`(s0^~s1|s1&s2)^s2`
		`(s0^~s1|~s0&s2)^s2`
		`(s0^~s2|s1&s2)^s1`
		`(s0^~s2|~s0&s1)^s1`
		`s0^(~s0&s1|~s2)^s1`
		`s0^(~s0&s2|~s1)^s2`
		`s0^(s0|~s1)&s2^~s1`
		`s0^(s0|~s2)&s1^~s2`
		`(s0|s1&s2)^s1^~s2`
		`(s0|s1&s2)^~s1^s2`
		`~s0&(~s1|~s2)^s1^s2`
		`~s0&(~s1|~s2)^~s1^~s2`
		`s0&~s1^(s0|~s1)&~s2`
		`s0&~s2^(s0|~s2)&~s1`
		`s0&(~s1|~s2)^~s1&~s2`
		`(s0|~s1)&(s0^s2)^~s1`
		`(s0|~s2)&(s0^s1)^~s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		(s0^s1)&(~s1|~s2)^~s2 (s0^s2)&(~s1|~s2)^~s1 (s0|~s1)&((~s0|s1)^s2) (s0|~s1)&(s0^s1^~s2) (s0|~s1)&(s0^~s1^s2) (s0|~s1)&(s0&~s1^~s2) (s0|~s2)&((~s0|s2)^s1) (s0|~s2)&(s0^s1^~s2) (s0|~s2)&(s0^~s1^s2) (s0|~s2)&(s0&~s2^~s1) (s0^~s1&~s2)&(~s1|~s2) (s0^s1^~s2)&(~s1|~s2) (s0^~s1^s2)&(~s1|~s2)	
00101010b	Boolean Function 0x2A	This symbol will compute s0&(~s1|~s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
00101011b	Boolean Function 0x2B	This symbol will compute the following functions: s0&~s1|(s0|~s1)&~s2 s0&~s1|(s0^~s1)&~s2 s0&~s2|(s0|~s2)&~s1 s0&~s2|(s0^~s2)&~s1 s0&(~s1|~s2)|~s1&~s2 s0&(s1^s2)|~s1&~s2 s0^(s0^~s1)&(s1^~s2) s0^(s0^~s2)&(s1^~s2) (s0|~s1)^(s0^~s1)&s2 (s0|~s2)^(s0^~s2)&s1 (s0|s1^~s2)^s1&s2 (~s0|s1)^(s0^s1|s2) (~s0|s2)^(s0^s2|s1) (~s0|s1^~s2)^(s1|s2) (s0^s1|s1^~s2)^s1 (s0^s2|s1^~s2)^s2 (s0^~s1|s1^~s2)^s2 (s0^~s2|s1^~s2)^s1 s0&~s1^(s0^~s1)&~s2 s0&~s2^(s0^~s2)&~s1 s0&(s1^s2)^~s1&~s2 ~s0&s1^(s0^s1|~s2) ~s0&s2^(s0^s2|~s1) ~s0&(s1^s2)^(~s1|~s2) (s0^s1)&(s1^s2)^~s2 (s0^s2)&(s1^s2)^~s1 (s0^~s1)&(s1^s2)^~s1 (s0^~s2)&(s1^s2)^~s2 (s0|~s1)&(s0^s1|~s2)	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		(s0|~s1)&(s0&~s1|~s2) (s0|~s2)&(s0^s2|~s1) (s0|~s2)&(s0&~s2|~s1) (s0|s1^~s2)&(~s1|~s2) (s0|~s1&~s2)&(~s1|~s2)	
00101100b	Boolean Function 0x2C	This symbol will compute the following functions: (~s0&s2|s1)^s2 (~s0&~s1|~s2)^~s1 (s0|s1)&s2^s1 (s0|~s2)&~s1^~s2 (s0|s1)&(s1^s2) (s0|~s2)&(s1^s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
00101101b	Boolean Function 0x2D	This symbol will compute the following functions: (~s0|s1)^s2 s0&~s1^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
00101110b	Boolean Function 0x2E	This symbol will compute the following functions: s0&~s1|s1&~s2 s0^(s0^~s2)&s1 (s0|s1)^s1&s2 (~s0|s1)^(~s1|s2) (s0^s2|s1)^s2 s0&~s1^s1&~s2 ~s0&~s1^(~s1|~s2) (s0^~s2)&~s1^~s2 (s0|s1)&(~s1|~s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
00101111b	Boolean Function 0x2F	This symbol will compute s0&~s1|~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
00110000b	Boolean Function 0x30	This symbol will compute ~s1&s2	Syntax: TERNARY_REGIMM_REG_REGIMM
00110001b	Boolean Function 0x31	This symbol will compute (~s0|s2)&~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
00110010b	Boolean Function 0x32	This symbol will compute (s0|s2)&~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
00110011b	Boolean Function 0x33	This symbol will compute ~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
00110100b	Boolean Function 0x34	This symbol will compute the following functions: (s0&s1|s2)^s1 (s0&~s2|~s1)^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl			
		(~s0|s2)&s1^s2 (~s0|~s1)&~s2^~s1 (~s0|s2)&(s1^s2) (~s0|~s1)&(s1^s2)	
00110101b	Boolean Function 0x35	This symbol will compute the following functions: ~s0&~s2|~s1&s2 s0^(s0^~s1|~s2) (s0|s2)^(s1|~s2) (~s0|s2)^s1&s2 (s0^~s1|s2)^s1 s0&~s2^(~s1|~s2) ~s0&~s2^~s1&s2 (s0^s1)&~s2^~s1 (~s0|s2)&(~s1|~s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
00110110b	Boolean Function 0x36	This symbol will compute the following functions: (s0|s2)^s1 ~s0&~s2^~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
00110111b	Boolean Function 0x37	This symbol will compute ~s0&~s2|~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
00111000b	Boolean Function 0x38	This symbol will compute the following functions: (~s0&s1|s2)^s1 (~s0&~s2|~s1)^~s2 (s0|s2)&s1^s2 (s0|~s1)&~s2^~s1 (s0|s2)&(s1^s2) (s0|~s1)&(s1^s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
00111001b	Boolean Function 0x39	This symbol will compute the following functions: (~s0|s2)^s1 s0&~s2^~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
00111010b	Boolean Function 0x3A	This symbol will compute the following functions: s0&~s2|~s1&s2 s0^(s0^~s1)&s2 (s0|s2)^s1&s2 (~s0|s2)^(s1|~s2) (s0^s1|s2)^s1 s0&~s2^~s1&s2 ~s0&~s2^(~s1|~s2) (s0^~s1)&~s2^~s1	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		`(s0|s2)&(~s1|~s2)`	
00111011b	Boolean Function 0x3B	This symbol will compute `s0&~s2|~s1`	Syntax: TERNARY_REGIMM_REG_REGIMM
00111100b	Boolean Function 0x3C	This symbol will compute `s1^s2`	Syntax: TERNARY_REGIMM_REG_REGIMM

00111101b	Boolean Function 0x3D	This symbol will compute the following functions:
		`~s0&~s1|s1^s2`
		`~s0&~s2|s1^s2`
		`(~s0&~s1|s2)^s1`
		`(~s0&~s2|s1)^s2`
		`(s0|s1)&~s2^~s1`
		`(s0|s2)&~s1^~s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

00111110b	Boolean Function 0x3E	This symbol will compute the following functions:
		`s0&~s1|s1^s2`
		`s0&~s2|s1^s2`
		`(s0&~s1|s2)^s1`
		`(s0&~s2|s1)^s2`
		`(~s0|s1)&~s2^~s1`
		`(~s0|s2)&~s1^~s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

00111111b	Boolean Function 0x3F	This symbol will compute `~s1|~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
01000000b	Boolean Function 0x40	This symbol will compute `~s0&s1&s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
01000001b	Boolean Function 0x41	This symbol will compute `~s0&(s1^~s2)`	Syntax: TERNARY_REGIMM_REG_REGIMM

01000010b	Boolean Function 0x42	This symbol will compute the following functions:
		`(s0^s1)&(s1^~s2)`
		`(s0^s2)&(s1^~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

01000011b	Boolean Function 0x43	This symbol will compute the following functions:
		`(s0&s1|~s2)^s1`
		`(s0&s2|~s1)^s2`
		`(~s0|~s1)&s2^~s1`
		`(~s0|~s2)&s1^~s2`
		`(~s0|~s1)&(s1^~s2)`
		`(~s0|~s2)&(s1^~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

01000100b	Boolean Function 0x44	This symbol will compute `~s0&s1`	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

01000101b	Boolean Function 0x45	This symbol will compute `~s0&(s1|~s2)`	Syntax: TERNARY_REGIMM_REG_REGIMM

01000110b	Boolean Function 0x46	This symbol will compute the following functions:
		`s0^(s0&s2|s1)`
		`(~s0|~s1&s2)^~s1`
		`s0&(s1|~s2)^s1`
		`(s0^s1)&(s1|~s2)`
		`(s0^s1)&(~s0|~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

01000111b	Boolean Function 0x47	This symbol will compute the following functions:
		`~s0&s1|~s1&~s2`
		`s0^(s0^~s2|s1)`
		`(s0|~s1)^(s1|s2)`
		`(~s0|~s1)^~s1&s2`
		`(s0^~s2|~s1)^s2`
		`s0&s1^(s1|~s2)`
		`~s0&s1^~s1&~s2`
		`(s0^s2)&s1^~s2`
		`(~s0|~s1)&(s1|~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

01001000b	Boolean Function 0x48	This symbol will compute `(s0^s2)&s1`	Syntax: TERNARY_REGIMM_REG_REGIMM

01001001b	Boolean Function 0x49	This symbol will compute the following functions:
		`s0^(s0&s2|s1^~s2)`
		`s0^(s0&~s1|s1^~s2)`
		`(s0|s2)^(s0&s2|~s1)`
		`(s0|~s1)^(s0&~s1|s2)`
		`(s0|~s1&s2)^(~s1|s2)`
		`(s0^s2|s0&~s1)^~s1`
		`(s0^s2|~s1&s2)^~s1`
		`(s0^~s1|s0&s2)^s2`
		`(s0^~s1|~s1&s2)^s2`
		`s0^(s0&s2|s1)^~s2`
		`s0^(s0&~s1|~s2)^s1`
		`s0^(~s0|s1)&s2^~s1`
		`s0^(~s0|~s2)&~s1^s2`
		`(~s0|~s1&s2)^s1^s2`
		`(~s0|~s1&s2)^~s1^~s2`
		`s0&(s1|~s2)^s1^~s2`
		`s0&(s1|~s2)^~s1^s2`
		`~s0&s1^(~s0|s1)&~s2`
		`~s0&~s2^(~s0|~s2)&s1`
		`~s0&(s1|~s2)^s1&~s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		(s0^s1)&(s1|~s2)^~s2 (s0^s1)&(~s0|~s2)^~s2 (s0^~s2)&(s1|~s2)^s1 (s0^~s2)&(~s0|s1)^s1 (~s0|s1)&((s0|~s1)^s2) (~s0|s1)&(s0^s1^~s2) (~s0|s1)&(s0^~s1^s2) (~s0|s1)&(~s0&s1^~s2) (~s0|~s2)&((s0|s2)^~s1) (~s0|~s2)&(s0^s1^~s2) (~s0|~s2)&(s0^~s1^s2) (~s0|~s2)&(~s0&~s2^s1) (s0^(~s1|s2))&(s1|~s2) (s0^s1^~s2)&(s1|~s2) (s0^~s1^s2)&(s1|~s2)	
01001010b	Boolean Function 0x4A	This symbol will compute the following functions: s0^(s0|s1)&s2 (s0|~s1&s2)^s2 ~s0&(s1|~s2)^~s2 (s0|s1)&(s0^s2) (s0^s2)&(s1|~s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
01001011b	Boolean Function 0x4B	This symbol will compute the following functions: (s0|~s1)^s2 ~s0&s1^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
01001100b	Boolean Function 0x4C	This symbol will compute (~s0|~s2)&s1	Syntax: TERNARY_REGIMM_REG_REGIMM
01001101b	Boolean Function 0x4D	This symbol will compute the following functions: ~s0&s1|(~s0|s1)&~s2 ~s0&s1|(s0^~s1)&~s2 ~s0&~s2|(~s0|~s2)&s1 ~s0&~s2|(s0^s2)&s1 ~s0&(s1|~s2)|s1&~s2 ~s0&(s1^~s2)|s1&~s2 s0^(s0^s1|s1^~s2) s0^(s0^~s2|s1^~s2) (s0|s2)^(s0^~s2|~s1) (s0|~s1)^(s0^s1|s2) (s0|s1^s2)^(~s1|s2) (~s0|s1)^(s0^~s1)&s2 (~s0|~s2)^(s0^s2)&~s1 (~s0|s1^s2)^~s1&s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		(s0^s1|s1^s2)^~s1 (s0^s2|s1^s2)^~s1 (s0^~s1|s1^s2)^s2 (s0^~s2|s1^s2)^s2 s0&s2^(s0^~s2|s1) s0&~s1^(s0^s1|~s2) s0&(s1^~s2)^(s1|~s2) ~s0&s1^(s0^~s1)&~s2 ~s0&~s2^(s0^s2)&s1 ~s0&(s1^~s2)^s1&~s2 (s0^s1)&(s1^~s2)^~s2 (s0^s2)&(s1^~s2)^~s2 (s0^~s1)&(s1^~s2)^s1 (s0^~s2)&(s1^~s2)^s1 (~s0|s1)&(s0^s1|~s2) (~s0|s1)&(~s0&s1|~s2) (~s0|~s2)&(s0^~s2|s1) (~s0|~s2)&(~s0&~s2|s1) (~s0|s1^s2)&(s1|~s2) (~s0|s1&~s2)&(s1|~s2)	
01001110b	Boolean Function 0x4E	This symbol will compute the following functions: s0&~s2|~s0&s1 (s0|s1)^s0&s2 (s0|~s1)^(~s0|s2) (s0|s1^s2)^s2 (~s0|~s2)^~s0&~s1 (~s0|s1^s2)^~s1 s0&~s2^~s0&s1 s0&(s1^~s2)^s1 ~s0&(s1^~s2)^~s2 (s0|s1)&(~s0|~s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
01001111b	Boolean Function 0x4F	This symbol will compute ~s0&s1|~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
01010000b	Boolean Function 0x50	This symbol will compute ~s0&s2	Syntax: TERNARY_REGIMM_REG_REGIMM
01010001b	Boolean Function 0x51	This symbol will compute ~s0&(~s1|s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
01010010b	Boolean Function 0x52	This symbol will compute the following functions: s0^(s0&s1|s2) (~s0|s1&~s2)^~s2 s0&(~s1|s2)^s2 (s0^s2)&(~s0|~s1)	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		`(s0^s2)&(~s1|s2)`	

01010011b	Boolean Function 0x53	This symbol will compute the following functions:
		`~s0&s2|~s1&~s2`
		`s0^(s0^~s1|s2)`
		`(s0|~s2)^(s1|s2)`
		`(~s0|~s2)^s1&~s2`
		`(s0^~s1|~s2)^s1`
		`s0&s2^(~s1|s2)`
		`~s0&s2^~s1&~s2`
		`(s0^s1)&s2^~s1`
		`(~s0|~s2)&(~s1|s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

01010100b	Boolean Function 0x54	This symbol will compute `~s0&(s1|s2)`	Syntax: TERNARY_REGIMM_REG_REGIMM
01010101b	Boolean Function 0x55	This symbol will compute `~s0`	Syntax: TERNARY_REGIMM_REG_REGIMM
01010110b	Boolean Function 0x56	This symbol will compute `s0^(s1|s2)`	Syntax: TERNARY_REGIMM_REG_REGIMM
01010111b	Boolean Function 0x57	This symbol will compute `~s0|~s1&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM

01011000b	Boolean Function 0x58	This symbol will compute the following functions:
		`s0^(s0&~s1|s2)`
		`(~s0|~s1&~s2)^~s2`
		`s0&(s1|s2)^s2`
		`(s0^s2)&(s1|s2)`
		`(s0^s2)&(~s0|s1)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

01011001b	Boolean Function 0x59	This symbol will compute `s0^(~s1|s2)`	Syntax: TERNARY_REGIMM_REG_REGIMM
01011010b	Boolean Function 0x5A	This symbol will compute `s0^s2`	Syntax: TERNARY_REGIMM_REG_REGIMM

01011011b	Boolean Function 0x5B	This symbol will compute the following functions:
		`s0^s2|~s0&~s1`
		`s0^s2|~s1&~s2`
		`s0^(~s0&~s1|s2)`
		`(s0|~s1&~s2)^s2`
		`~s0&(s1|s2)^~s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

01011100b	Boolean Function 0x5C	This symbol will compute the following functions: `~s0&s2|s1&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		s0^(s0^s1|s2) (s0|~s2)^(~s1|s2) (~s0|~s2)^~s1&~s2 (s0^s1|~s2)^~s1 s0&s2^(s1|s2) ~s0&s2^s1&~s2 (s0^~s1)&s2^s1 (~s0|~s2)&(s1|s2)	
01011101b	Boolean Function 0x5D	This symbol will compute ~s0|s1&~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
01011110b	Boolean Function 0x5E	This symbol will compute the following functions: s0^s2|s1&~s2 s0^s2|~s0&s1 s0^(~s0&s1|s2) (s0|s1&~s2)^s2 ~s0&(~s1|s2)^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
01011111b	Boolean Function 0x5F	This symbol will compute ~s0|~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
01100000b	Boolean Function 0x60	This symbol will compute (s0^s1)&s2	Syntax: TERNARY_REGIMM_REG_REGIMM
01100001b	Boolean Function 0x61	This symbol will compute the following functions: s0^(s0&s1|s1^~s2) s0^(s0&~s2|s1^~s2) (s0|s1)^(s0&s1|~s2) (s0|~s2)^(s0&~s2|s1) (s0|s1&~s2)^(s1|~s2) (s0^s1|s0&~s2)^~s2 (s0^s1|s1&~s2)^~s2 (s0^~s2|s0&s1)^s1 (s0^~s2|s1&~s2)^s1 s0^(s0&s1|s2)^~s1 s0^(s0&~s2|~s1)^s2 s0^(~s0|s2)&s1^~s2 s0^(~s0|~s1)&~s2^s1 (~s0|s1&~s2)^s1^s2 (~s0|s1&~s2)^~s1^~s2 s0&(~s1|s2)^s1^~s2 s0&(~s1|s2)^~s1^s2 ~s0&s2^(~s0|s2)&~s1 ~s0&~s1^(~s0|~s1)&s2 ~s0&(~s1|s2)^~s1&s2 (s0^s2)&(~s0|~s1)^~s1	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		(s0^s2)&(~s1|s2)^~s1 (s0^~s1)&(~s0|s2)^s2 (s0^~s1)&(~s1|s2)^s2 (~s0|s2)&((s0|~s2)^s1) (~s0|s2)&(s0^s1^~s2) (~s0|s2)&(s0^~s1^s2) (~s0|s2)&(~s0&s2^~s1) (~s0|~s1)&((s0|s1)^~s2) (~s0|~s1)&(s0^s1^~s2) (~s0|~s1)&(s0^~s1^s2) (~s0|~s1)&(~s0&~s1^s2) (s0^(s1|~s2))&(~s1|s2) (s0^s1^~s2)&(~s1|s2) (s0^~s1^s2)&(~s1|s2)	
01100010b	Boolean Function 0x62	This symbol will compute the following functions: s0^(s0|s2)&s1 (s0|s1&~s2)^s1 ~s0&(~s1|s2)^~s1 (s0|s2)&(s0^s1) (s0^s1)&(~s1|s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
01100011b	Boolean Function 0x63	This symbol will compute the following functions: (s0|~s2)^s1 ~s0&s2^~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
01100100b	Boolean Function 0x64	This symbol will compute the following functions: s0^(s0&~s2|s1) (~s0|~s1&~s2)^~s1 s0&(s1|s2)^s1 (s0^s1)&(s1|s2) (s0^s1)&(~s0|s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
01100101b	Boolean Function 0x65	This symbol will compute s0^(s1|~s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
01100110b	Boolean Function 0x66	This symbol will compute s0^s1	Syntax: TERNARY_REGIMM_REG_REGIMM
01100111b	Boolean Function 0x67	This symbol will compute the following functions: s0^s1|~s0&~s2 s0^s1|~s1&~s2 s0^(~s0&~s2|s1) (s0|~s1&~s2)^s1 ~s0&(s1|s2)^~s1	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

01101000b	Boolean Function 0x68	This symbol will compute the following functions:
		`s0^(s0|s1)&(s1^~s2)`
		`s0^(s0|s2)&(s1^~s2)`
		`(~s0|~s1)^(~s0&~s1|~s2)`
		`(~s0|~s2)^(~s0&~s2|~s1)`
		`(~s0|~s1&~s2)^(~s1|~s2)`
		`(s0^s1|~s0&~s2)^~s2`
		`(s0^s1|~s1&~s2)^~s2`
		`(s0^s2|~s0&~s1)^~s1`
		`(s0^s2|~s1&~s2)^~s1`
		`s0^(~s0&~s1|s2)^~s1`
		`s0^(~s0&~s2|s1)^~s2`
		`s0^(s0|s1)&~s2^s1`
		`s0^(s0|s2)&~s1^s2`
		`(s0|~s1&~s2)^s1^~s2`
		`(s0|~s1&~s2)^~s1^s2`
		`~s0&(s1|s2)^s1^s2`
		`~s0&(s1|s2)^~s1^~s2`
		`s0&s1^(s0|s1)&s2`
		`s0&s2^(s0|s2)&s1`
		`s0&(s1|s2)^s1&s2`
		`(s0|s1)&(s0^~s2)^s1`
		`(s0|s2)&(s0^~s1)^s2`
		`(s0^~s1)&(s1|s2)^s2`
		`(s0^~s2)&(s1|s2)^s1`
		`(s0|s1)&((~s0|~s1)^~s2)`
		`(s0|s1)&(s0^s1^~s2)`
		`(s0|s1)&(s0^~s1^s2)`
		`(s0|s1)&(s0&s1^s2)`
		`(s0|s2)&((~s0|~s2)^~s1)`
		`(s0|s2)&(s0^s1^~s2)`
		`(s0|s2)&(s0^~s1^s2)`
		`(s0|s2)&(s0&s2^s1)`
		`(s0^s1&s2)&(s1|s2)`
		`(s0^s1^~s2)&(s1|s2)`
		`(s0^~s1^s2)&(s1|s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

01101001b	Boolean Function 0x69	This symbol will compute the following functions:
		`s0^s1^~s2`
		`s0^~s1^s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

01101010b	Boolean Function 0x6A	This symbol will compute `s0^s1&s2`	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl			

01101011b	Boolean Function 0x6B	This symbol will compute the following functions:
		`s0^s1&s2|~s1&~s2`
		`s0^s1^~s2|~s1&~s2`
		`s0^~s1^s2|~s1&~s2`
		`s0&~s1|(s0|~s1)^s2`
		`s0&~s1|s0^s1^~s2`
		`s0&~s1|s0^~s1^s2`
		`s0&~s1|~s0&s1^~s2`
		`s0&~s2|(s0|~s2)^s1`
		`s0&~s2|s0^s1^~s2`
		`s0&~s2|s0^~s1^s2`
		`s0&~s2|~s0&s2^~s1`
		`s0^(~s0|s1)&(s1^~s2)`
		`s0^(~s0|s2)&(s1^~s2)`
		`(s0|~s1)^(~s0|s1)&s2`
		`(s0|~s2)^(~s0|s2)&s1`
		`(s0|~s1&~s2)^s1&s2`
		`(s0^~s1|s0&~s2)^s2`
		`(s0^~s1|~s1&~s2)^s2`
		`(s0^~s2|s0&~s1)^s1`
		`(s0^~s2|~s1&~s2)^s1`
		`s0^(s0&~s1|s2)^~s1`
		`s0^(s0&~s2|s1)^~s2`
		`s0^(~s0|s1)&~s2^s1`
		`s0^(~s0|s2)&~s1^s2`
		`(~s0|~s1&~s2)^s1^s2`
		`(~s0|~s1&~s2)^~s1^~s2`
		`s0&(s1|s2)^s1^~s2`
		`s0&(s1|s2)^~s1^s2`
		`~s0&s1^(s0&~s1|~s2)`
		`~s0&s2^(s0&~s2|~s1)`
		`~s0&(s1|s2)^(~s1|~s2)`
		`(s0^s1)&(s1|s2)^~s2`
		`(s0^s1)&(~s0|s2)^~s2`
		`(s0^s2)&(s1|s2)^~s1`
		`(s0^s2)&(~s0|s1)^~s1`
			Syntax: TERNARY_REGIMM_REG_REGIMM

01101100b	Boolean Function 0x6C	This symbol will compute the following functions:
		`(~s0|~s2)^~s1`
		`s0&s2^s1`
			Syntax: TERNARY_REGIMM_REG_REGIMM

01101101b	Boolean Function 0x6D	This symbol will compute the following functions:
		`s0^(s1|~s2)|s1&~s2`
		`s0^s1^~s2|s1&~s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		s0^~s1^s2|s1&~s2 ~s0&s1|(~s0|s1)^s2 ~s0&s1|s0^s1^~s2 ~s0&s1|s0^~s1^s2 ~s0&s1|s0&~s1^~s2 ~s0&~s2|(~s0|~s2)^~s1 ~s0&~s2|s0^s1^~s2 ~s0&~s2|s0^~s1^s2 ~s0&~s2|s0&s2^s1 s0^(~s0&s1|s1^~s2) s0^(~s0&~s2|s1^~s2) (~s0|s1)^(s0|~s1)&s2 (~s0|~s2)^(s0|s2)&~s1 (~s0|s1&~s2)^~s1&s2 (s0^s2|s1&~s2)^~s1 (s0^s2|~s0&s1)^~s1 (s0^~s1|s1&~s2)^s2 (s0^~s1|~s0&~s2)^s2 s0^(~s0&s1|s2)^~s1 s0^(~s0&~s2|~s1)^s2 s0^(s0|s2)&s1^~s2 s0^(s0|~s1)&~s2^s1 (s0|s1&~s2)^s1^~s2 (s0|s1&~s2)^~s1^s2 ~s0&(~s1|s2)^s1^s2 ~s0&(~s1|s2)^~s1^~s2 s0&s2^(~s0&~s2|s1) s0&~s1^(~s0&s1|~s2) s0&(~s1|s2)^(s1|~s2) (s0|s2)&(s0^s1)^~s2 (s0|~s1)&(s0^~s2)^s1 (s0^s1)&(~s1|s2)^~s2 (s0^~s2)&(~s1|s2)^s1	
01101110b	Boolean Function 0x6E	This symbol will compute the following functions: s0^s1|s0&~s2 s0^s1|s1&~s2 s0^(~s0|s2)&s1 (~s0|s1&~s2)^~s1 s0&(~s1|s2)^s1	Syntax: TERNARY_REGIMM_REG_REGIMM
01101111b	Boolean Function 0x6F	This symbol will compute s0^s1|~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
01110000b	Boolean Function 0x70	This symbol will compute (~s0|~s1)&s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

01110001b	Boolean Function 0x71	This symbol will compute the following functions:
		`~s0&s2|(~s0|s2)&~s1`
		`~s0&s2|(s0^~s2)&~s1`
		`~s0&~s1|(~s0|~s1)&s2`
		`~s0&~s1|(s0^s1)&s2`
		`~s0&(~s1|s2)|~s1&s2`
		`~s0&(s1^~s2)|~s1&s2`
		`s0^(s0^s2|s1^~s2)`
		`s0^(s0^~s1|s1^~s2)`
		`(s0|s1)^(s0^~s1|~s2)`
		`(s0|~s2)^(s0^s2|s1)`
		`(s0|s1^s2)^(s1|~s2)`
		`(~s0|s2)^(s0^~s2)&s1`
		`(~s0|~s1)^(s0^s1)&~s2`
		`(~s0|s1^s2)^s1&~s2`
		`(s0^s1|s1^s2)^~s2`
		`(s0^s2|s1^s2)^~s2`
		`(s0^~s1|s1^s2)^s1`
		`(s0^~s2|s1^s2)^s1`
		`s0&s1^(s0^~s1|s2)`
		`s0&~s2^(s0^s2|~s1)`
		`s0&(s1^~s2)^(~s1|s2)`
		`~s0&s2^(s0^~s2)&~s1`
		`~s0&~s1^(s0^s1)&s2`
		`~s0&(s1^~s2)^~s1&s2`
		`(s0^s1)&(s1^~s2)^~s1`
		`(s0^s2)&(s1^~s2)^~s1`
		`(s0^~s1)&(s1^~s2)^s2`
		`(s0^~s2)&(s1^~s2)^s2`
		`(~s0|s2)&(s0^s2|~s1)`
		`(~s0|s2)&(~s0&s2|~s1)`
		`(~s0|~s1)&(s0^~s1|s2)`
		`(~s0|~s1)&(~s0&~s1|s2)`
		`(~s0|s1^s2)&(~s1|s2)`
		`(~s0|~s1&s2)&(~s1|s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

01110010b	Boolean Function 0x72	This symbol will compute the following functions:
		`s0&~s1|~s0&s2`
		`(s0|s2)^s0&s1`
		`(s0|~s2)^(~s0|s1)`
		`(s0|s1^s2)^s1`
		`(~s0|~s1)^~s0&~s2`
		`(~s0|s1^s2)^~s2`
		`s0&~s1^~s0&s2`
		`s0&(s1^~s2)^s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		~s0&(s1^~s2)^~s1 (s0|s2)&(~s0|~s1)	
01110011b	Boolean Function 0x73	This symbol will compute ~s0&s2|~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
01110100b	Boolean Function 0x74	This symbol will compute the following functions: ~s0&s1|~s1&s2 s0^(s0^s2|s1) (s0|~s1)^(s1|~s2) (~s0|~s1)^~s1&~s2 (s0^s2|~s1)^~s2 s0&s1^(s1|s2) ~s0&s1^~s1&s2 (s0^~s2)&s1^s2 (~s0|~s1)&(s1|s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
01110101b	Boolean Function 0x75	This symbol will compute ~s0|~s1&s2	Syntax: TERNARY_REGIMM_REG_REGIMM
01110110b	Boolean Function 0x76	This symbol will compute the following functions: s0^s1|~s0&s2 s0^s1|~s1&s2 s0^(~s0&s2|s1) (s0|~s1&s2)^s1 ~s0&(s1|~s2)^~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
01110111b	Boolean Function 0x77	This symbol will compute ~s0|~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
01111000b	Boolean Function 0x78	This symbol will compute the following functions: (~s0|~s1)^~s2 s0&s1^s2	Syntax: TERNARY_REGIMM_REG_REGIMM
01111001b	Boolean Function 0x79	This symbol will compute the following functions: s0^(~s1|s2)|~s1&s2 s0^s1^~s2|~s1&s2 s0^~s1^s2|~s1&s2 ~s0&s2|(~s0|s2)^s1 ~s0&s2|s0^s1^~s2 ~s0&s2|s0^~s1^s2 ~s0&s2|s0&~s2^~s1 ~s0&~s1|(~s0|~s1)^~s2 ~s0&~s1|s0^s1^~s2 ~s0&~s1|s0^~s1^s2 ~s0&~s1|s0&s1^s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		s0^(~s0&s2|s1^~s2) s0^(~s0&~s1|s1^~s2) (~s0|s2)^(s0|~s2)&s1 (~s0|~s1)^(s0|s1)&~s2 (~s0|~s1&s2)^s1&~s2 (s0^s1|~s0&s2)^~s2 (s0^s1|~s1&s2)^~s2 (s0^~s2|~s0&~s1)^s1 (s0^~s2|~s1&s2)^s1 s0^(~s0&s2|s1)^~s2 s0^(~s0&~s1|~s2)^s1 s0^(s0|s1)&s2^~s1 s0^(s0|~s2)&~s1^s2 (s0|~s1&s2)^s1^~s2 (s0|~s1&s2)^~s1^s2 ~s0&(s1|~s2)^s1^s2 ~s0&(s1|~s2)^~s1^~s2 s0&s1^(~s0&~s1|s2) s0&~s2^(~s0&s2|~s1) s0&(s1|~s2)^(~s1|s2) (s0|s1)&(s0^s2)^~s1 (s0|~s2)&(s0^~s1)^s2 (s0^s2)&(s1|~s2)^~s1 (s0^~s1)&(s1|~s2)^s2	
01111010b	Boolean Function 0x7A	This symbol will compute the following functions: s0^s2|s0&~s1 s0^s2|~s1&s2 s0^(~s0|s1)&s2 (~s0|~s1&s2)^~s2 s0&(s1|~s2)^s2	Syntax: TERNARY_REGIMM_REG_REGIMM
01111011b	Boolean Function 0x7B	This symbol will compute s0^s2|~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
01111100b	Boolean Function 0x7C	This symbol will compute the following functions: ~s0&s1|s1^s2 ~s0&s2|s1^s2 (~s0&s1|~s2)^~s1 (~s0&s2|~s1)^~s2 (s0|~s1)&s2^s1 (s0|~s2)&s1^s2	Syntax: TERNARY_REGIMM_REG_REGIMM
01111101b	Boolean Function 0x7D	This symbol will compute ~s0|s1^s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl			

01111110b	Boolean Function 0x7E	This symbol will compute the following functions:
		`s0^s1|s1^s2`
		`s0^s2|s1^s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

01111111b	Boolean Function 0x7F	This symbol will compute `~s0|~s1|~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
10000000b	Boolean Function 0x80	This symbol will compute `s0&s1&s2`	Syntax: TERNARY_REGIMM_REG_REGIMM

10000001b	Boolean Function 0x81	This symbol will compute the following functions:
		`(s0^~s1)&(s1^~s2)`
		`(s0^~s2)&(s1^~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

10000010b	Boolean Function 0x82	This symbol will compute `s0&(s1^~s2)`	Syntax: TERNARY_REGIMM_REG_REGIMM

10000011b	Boolean Function 0x83	This symbol will compute the following functions:
		`(~s0&s1|~s2)^s1`
		`(~s0&s2|~s1)^s2`
		`(s0|~s1)&s2^~s1`
		`(s0|~s2)&s1^~s2`
		`(s0|~s1)&(s1^~s2)`
		`(s0|~s2)&(s1^~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

10000100b	Boolean Function 0x84	This symbol will compute `(s0^~s2)&s1`	Syntax: TERNARY_REGIMM_REG_REGIMM

10000101b	Boolean Function 0x85	This symbol will compute the following functions:
		`s0^(s0&~s1|~s2)`
		`(~s0|~s1&s2)^s2`
		`s0&(s1|~s2)^~s2`
		`(s0^~s2)&(s1|~s2)`
		`(s0^~s2)&(~s0|s1)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

10000110b	Boolean Function 0x86	This symbol will compute the following functions:
		`s0^(s0|s1)&(s1^s2)`
		`s0^(s0|~s2)&(s1^s2)`
		`(~s0|s2)^(~s0&s2|~s1)`
		`(~s0|~s1)^(~s0&~s1|s2)`
		`(~s0|~s1&s2)^(~s1|s2)`
		`(s0^s1|~s0&s2)^s2`
		`(s0^s1|~s1&s2)^s2`
		`(s0^~s2|~s0&~s1)^~s1`
		`(s0^~s2|~s1&s2)^~s1`
			Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		s0^(~s0&s2|s1)^s2 s0^(~s0&~s1|~s2)^~s1 s0^(s0|s1)&s2^s1 s0^(s0|~s2)&~s1^~s2 (s0|~s1&s2)^s1^s2 (s0|~s1&s2)^~s1^~s2 ~s0&(s1|~s2)^s1^~s2 ~s0&(s1|~s2)^~s1^s2 s0&s1^(s0|s1)&~s2 s0&~s2^(s0|~s2)&s1 s0&(s1|~s2)^s1&~s2 (s0|s1)&(s0^s2)^s1 (s0|~s2)&(s0^~s1)^~s2 (s0^s2)&(s1|~s2)^s1 (s0^~s1)&(s1|~s2)^~s2 (s0|s1)&((~s0|~s1)^s2) (s0|s1)&(s0^s1^s2) (s0|s1)&(s0^~s1^~s2) (s0|s1)&(s0&s1^~s2) (s0|~s2)&((~s0|s2)^~s1) (s0|~s2)&(s0^s1^s2) (s0|~s2)&(s0^~s1^~s2) (s0|~s2)&(s0&~s2^s1) (s0^s1&~s2)&(s1|~s2) (s0^s1^s2)&(s1|~s2) (s0^~s1^~s2)&(s1|~s2)	
10000111b	Boolean Function 0x87	This symbol will compute the following functions: (~s0|~s1)^s2 s0&s1^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
10001000b	Boolean Function 0x88	This symbol will compute s0&s1	Syntax: TERNARY_REGIMM_REG_REGIMM
10001001b	Boolean Function 0x89	This symbol will compute the following functions: s0^(s0|~s2)&~s1 (s0|~s1&s2)^~s1 ~s0&(s1|~s2)^s1 (s0|~s2)&(s0^~s1) (s0^~s1)&(s1|~s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
10001010b	Boolean Function 0x8A	This symbol will compute s0&(s1|~s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
10001011b	Boolean Function 0x8B	This symbol will compute the following functions: s0&s1|~s1&~s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		s0^(s0^~s2)&~s1 (s0|~s1)^~s1&s2 (~s0|~s1)^(s1|s2) (s0^s2|~s1)^s2 s0&s1^~s1&~s2 ~s0&s1^(s1|~s2) (s0^~s2)&s1^~s2 (s0|~s1)&(s1|~s2)	
10001100b	Boolean Function 0x8C	This symbol will compute (s0|~s2)&s1	Syntax: TERNARY_REGIMM_REG_REGIMM
10001101b	Boolean Function 0x8D	This symbol will compute the following functions: s0&s1|~s0&~s2 (s0|s2)^(~s0|~s1) (s0|~s2)^s0&~s1 (s0|s1^s2)^~s1 (~s0|s1)^~s0&s2 (~s0|s1^s2)^s2 s0&s1^~s0&~s2 s0&(s1^~s2)^~s2 ~s0&(s1^~s2)^s1 (s0|~s2)&(~s0|s1)	Syntax: TERNARY_REGIMM_REG_REGIMM
10001110b	Boolean Function 0x8E	This symbol will compute the following functions: s0&s1|(s0|s1)&~s2 s0&s1|(s0^s1)&~s2 s0&~s2|(s0|~s2)&s1 s0&~s2|(s0^~s2)&s1 s0&(s1|~s2)|s1&~s2 s0&(s1^~s2)|s1&~s2 s0^(s0^s1)&(s1^s2) s0^(s0^~s2)&(s1^s2) (s0|s1)^(s0^s1)&s2 (s0|~s2)^(s0^~s2)&~s1 (s0|s1^s2)^~s1&s2 (~s0|s2)^(s0^s2|~s1) (~s0|~s1)^(s0^~s1|s2) (~s0|s1^s2)^(~s1|s2) (s0^s1|s1^s2)^s2 (s0^s2|s1^s2)^s2 (s0^~s1|s1^s2)^~s1 (s0^~s2|s1^s2)^~s1 s0&s1^(s0^s1)&~s2 s0&~s2^(s0^~s2)&s1 s0&(s1^~s2)^s1&~s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		~s0&s2^(s0^s2|s1) ~s0&~s1^(s0^~s1|~s2) ~s0&(s1^~s2)^(s1|~s2) (s0^s1)&(s1^~s2)^s1 (s0^s2)&(s1^~s2)^s1 (s0^~s1)&(s1^~s2)^~s2 (s0^~s2)&(s1^~s2)^~s2 (s0|s1)&(s0^~s1|~s2) (s0|s1)&(s0&s1|~s2) (s0|~s2)&(s0^s2|s1) (s0|~s2)&(s0&~s2|s1) (s0|s1^s2)&(s1|~s2) (s0|s1&~s2)&(s1|~s2)	
10001111b	Boolean Function 0x8F	This symbol will compute s0&s1|~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
10010000b	Boolean Function 0x90	This symbol will compute (s0^~s1)&s2	Syntax: TERNARY_REGIMM_REG_REGIMM
10010001b	Boolean Function 0x91	This symbol will compute the following functions: s0^(s0&~s2|~s1) (~s0|s1&~s2)^s1 s0&(~s1|s2)^~s1 (s0^~s1)&(~s0|s2) (s0^~s1)&(~s1|s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
10010010b	Boolean Function 0x92	This symbol will compute the following functions: s0^(s0|s2)&(s1^s2) s0^(s0|~s1)&(s1^s2) (~s0|s1)^(~s0&s1|~s2) (~s0|~s2)^(~s0&~s2|s1) (~s0|s1&~s2)^(s1|~s2) (s0^s2|s1&~s2)^s1 (s0^s2|~s0&s1)^s1 (s0^~s1|s1&~s2)^~s2 (s0^~s1|~s0&~s2)^~s2 s0^(~s0&s1|s2)^s1 s0^(~s0&~s2|~s1)^~s2 s0^(s0|s2)&s1^s2 s0^(s0|~s1)&~s2^~s1 (s0|s1&~s2)^s1^s2 (s0|s1&~s2)^~s1^~s2 ~s0&(~s1|s2)^s1^~s2 ~s0&(~s1|s2)^~s1^s2 s0&s2^(s0|s2)&~s1 s0&~s1^(s0|~s1)&s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		s0&(~s1|s2)^~s1&s2 (s0|s2)&(s0^s1)^s2 (s0|~s1)&(s0^~s2)^~s1 (s0^s1)&(~s1|s2)^s2 (s0^~s2)&(~s1|s2)^~s1 (s0|s2)&((~s0|~s2)^s1) (s0|s2)&(s0^s1^s2) (s0|s2)&(s0^~s1^~s2) (s0|s2)&(s0&s2^~s1) (s0|~s1)&((~s0|s1)^~s2) (s0|~s1)&(s0^s1^s2) (s0|~s1)&(s0^~s1^~s2) (s0|~s1)&(s0&~s1^s2) (s0^~s1&s2)&(~s1|s2) (s0^s1^s2)&(~s1|s2) (s0^~s1^~s2)&(~s1|s2)	
10010011b	Boolean Function 0x93	This symbol will compute the following functions: (~s0|~s2)^s1 s0&s2^~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
10010100b	Boolean Function 0x94	This symbol will compute the following functions: s0^(s0&~s1|s1^s2) s0^(s0&~s2|s1^s2) (s0|~s1)^(s0&~s1|~s2) (s0|~s2)^(s0&~s2|~s1) (s0|~s1&~s2)^(~s1|~s2) (s0^~s1|s0&~s2)^~s2 (s0^~s1|~s1&~s2)^~s2 (s0^~s2|s0&~s1)^~s1 (s0^~s2|~s1&~s2)^~s1 s0^(s0&~s1|s2)^s1 s0^(s0&~s2|s1)^s2 s0^(~s0|s1)&~s2^~s1 s0^(~s0|s2)&~s1^~s2 (~s0|~s1&~s2)^s1^~s2 (~s0|~s1&~s2)^~s1^s2 s0&(s1|s2)^s1^s2 s0&(s1|s2)^~s1^~s2 ~s0&s1^(~s0|s1)&s2 ~s0&s2^(~s0|s2)&s1 ~s0&(s1|s2)^s1&s2 (s0^s1)&(s1|s2)^s2 (s0^s1)&(~s0|s2)^s2 (s0^s2)&(s1|s2)^s1 (s0^s2)&(~s0|s1)^s1	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl			
		(~s0|s1)&((s0|~s1)^~s2) (~s0|s1)&(s0^s1^s2) (~s0|s1)&(s0^~s1^~s2) (~s0|s1)&(~s0&s1^s2) (~s0|s2)&((s0|~s2)^~s1) (~s0|s2)&(s0^s1^s2) (~s0|s2)&(s0^~s1^~s2) (~s0|s2)&(~s0&s2^s1) (s0^(~s1|~s2))&(s1|s2) (s0^s1^s2)&(s1|s2) (s0^~s1^~s2)&(s1|s2)	
10010101b	Boolean Function 0x95	This symbol will compute s0^(~s1|~s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
10010110b	Boolean Function 0x96	This symbol will compute the following functions: s0^s1^s2 s0^~s1^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
10010111b	Boolean Function 0x97	This symbol will compute the following functions: s0^(~s1|~s2)|~s1&~s2 s0^s1^s2|~s1&~s2 s0^~s1^~s2|~s1&~s2 ~s0&~s1|(~s0|~s1)^s2 ~s0&~s1|s0^s1^s2 ~s0&~s1|s0^~s1^~s2 ~s0&~s1|s0&s1^~s2 ~s0&~s2|(~s0|~s2)^s1 ~s0&~s2|s0^s1^s2 ~s0&~s2|s0^~s1^~s2 ~s0&~s2|s0&s2^~s1 s0^(~s0&~s1|s1^s2) s0^(~s0&~s2|s1^s2) (~s0|~s1)^(s0|s1)&s2 (~s0|~s2)^(s0|s2)&s1 (~s0|~s1&~s2)^s1&s2 (s0^s1|~s0&~s2)^s2 (s0^s1|~s1&~s2)^s2 (s0^s2|~s0&~s1)^s1 (s0^s2|~s1&~s2)^s1 s0^(~s0&~s1|s2)^s1 s0^(~s0&~s2|s1)^s2 s0^(s0|s1)&~s2^~s1 s0^(s0|s2)&~s1^~s2 (s0|~s1&~s2)^s1^s2 (s0|~s1&~s2)^~s1^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		~s0&(s1|s2)^s1^~s2 ~s0&(s1|s2)^~s1^s2 s0&s1^(~s0&~s1|~s2) s0&s2^(~s0&~s2|~s1) s0&(s1|s2)^(~s1|~s2) (s0|s1)&(s0^~s2)^~s1 (s0|s2)&(s0^~s1)^~s2 (s0^~s1)&(s1|s2)^~s2 (s0^~s2)&(s1|s2)^~s1	
10011000b	Boolean Function 0x98	This symbol will compute the following functions: s0^(s0|s2)&~s1 (s0|~s1&~s2)^~s1 ~s0&(s1|s2)^s1 (s0|s2)&(s0^~s1) (s0^~s1)&(s1|s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
10011001b	Boolean Function 0x99	This symbol will compute s0^~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
10011010b	Boolean Function 0x9A	This symbol will compute s0^~s1&s2	Syntax: TERNARY_REGIMM_REG_REGIMM
10011011b	Boolean Function 0x9B	This symbol will compute the following functions: s0^~s1|s0&~s2 s0^~s1|~s1&~s2 s0^(~s0|s2)&~s1 (~s0|~s1&~s2)^s1 s0&(s1|s2)^~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
10011100b	Boolean Function 0x9C	This symbol will compute the following functions: (s0|~s2)^~s1 ~s0&s2^s1	Syntax: TERNARY_REGIMM_REG_REGIMM
10011101b	Boolean Function 0x9D	This symbol will compute the following functions: s0^~s1|s1&~s2 s0^~s1|~s0&~s2 s0^(~s0&~s2|~s1) (s0|s1&~s2)^~s1 ~s0&(~s1|s2)^s1	Syntax: TERNARY_REGIMM_REG_REGIMM
10011110b	Boolean Function 0x9E	This symbol will compute the following functions: s0^~s1&s2|s1&~s2 s0^s1^s2|s1&~s2 s0^~s1^~s2|s1&~s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		s0&s1|(s0|s1)^s2 s0&s1|s0^s1^s2 s0&s1|s0^~s1^~s2 s0&s1|~s0&~s1^~s2 s0&~s2|(s0|~s2)^~s1 s0&~s2|s0^s1^s2 s0&~s2|s0^~s1^~s2 s0&~s2|~s0&s2^s1 s0^(~s0|s2)&(s1^s2) s0^(~s0|~s1)&(s1^s2) (s0|s1)^(~s0|~s1)&s2 (s0|~s2)^(~s0|s2)&~s1 (s0|s1&~s2)^~s1&s2 (s0^s1|s0&~s2)^s2 (s0^s1|s1&~s2)^s2 (s0^~s2|s0&s1)^~s1 (s0^~s2|s1&~s2)^~s1 s0^(s0&s1|s2)^s1 s0^(s0&~s2|~s1)^~s2 s0^(~s0|s2)&s1^s2 s0^(~s0|~s1)&~s2^~s1 (~s0|s1&~s2)^s1^~s2 (~s0|s1&~s2)^~s1^s2 s0&(~s1|s2)^s1^s2 s0&(~s1|s2)^~s1^~s2 ~s0&s2^(s0&~s2|s1) ~s0&~s1^(s0&s1|~s2) ~s0&(~s1|s2)^(s1|~s2) (s0^s2)&(~s0|~s1)^s1 (s0^s2)&(~s1|s2)^s1 (s0^~s1)&(~s0|s2)^~s2 (s0^~s1)&(~s1|s2)^~s2	
10011111b	Boolean Function 0x9F	This symbol will compute s0^~s1|~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
10100000b	Boolean Function 0xA0	This symbol will compute s0&s2	Syntax: TERNARY_REGIMM_REG_REGIMM
10100001b	Boolean Function 0xA1	This symbol will compute the following functions: s0^(s0|~s1)&~s2 (s0|s1&~s2)^~s2 ~s0&(~s1|s2)^s2 (s0|~s1)&(s0^~s2) (s0^~s2)&(~s1|s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
10100010b	Boolean Function 0xA2	This symbol will compute s0&(~s1|s2)	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

10100011b	Boolean Function 0xA3	This symbol will compute the following functions:
		`s0&s2|~s1&~s2`
		`s0^(s0^~s1)&~s2`
		`(s0|~s2)^s1&~s2`
		`(~s0|~s2)^(s1|s2)`
		`(s0^s1|~s2)^s1`
		`s0&s2^~s1&~s2`
		`~s0&s2^(~s1|s2)`
		`(s0^~s1)&s2^~s1`
		`(s0|~s2)&(~s1|s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

10100100b	Boolean Function 0xA4	This symbol will compute the following functions:
		`s0^(s0|s1)&~s2`
		`(s0|~s1&~s2)^~s2`
		`~s0&(s1|s2)^s2`
		`(s0|s1)&(s0^~s2)`
		`(s0^~s2)&(s1|s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

10100101b	Boolean Function 0xA5	This symbol will compute `s0^~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
10100110b	Boolean Function 0xA6	This symbol will compute `s0^s1&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM

10100111b	Boolean Function 0xA7	This symbol will compute the following functions:
		`s0^~s2|s0&~s1`
		`s0^~s2|~s1&~s2`
		`s0^(~s0|s1)&~s2`
		`(~s0|~s1&~s2)^s2`
		`s0&(s1|s2)^~s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

10101000b	Boolean Function 0xA8	This symbol will compute `s0&(s1|s2)`	Syntax: TERNARY_REGIMM_REG_REGIMM
10101001b	Boolean Function 0xA9	This symbol will compute `s0^~s1&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
10101010b	Boolean Function 0xAA	This symbol will compute `s0`	Syntax: TERNARY_REGIMM_REG_REGIMM
10101011b	Boolean Function 0xAB	This symbol will compute `s0|~s1&~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM

10101100b	Boolean Function 0xAC	This symbol will compute the following functions:
		`s0&s2|s1&~s2`
		`s0^(s0^s1)&~s2`
		`(s0|~s2)^~s1&~s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		(~s0|~s2)^(~s1|s2) (s0^~s1|~s2)^~s1 s0&s2^s1&~s2 ~s0&s2^(s1|s2) (s0^s1)&s2^s1 (s0|~s2)&(s1|s2)	
10101101b	Boolean Function 0xAD	This symbol will compute the following functions: s0^~s2|s0&s1 s0^~s2|s1&~s2 s0^(~s0|~s1)&~s2 (~s0|s1&~s2)^s2 s0&(~s1|s2)^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
10101110b	Boolean Function 0xAE	This symbol will compute s0|s1&~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
10101111b	Boolean Function 0xAF	This symbol will compute s0|~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
10110000b	Boolean Function 0xB0	This symbol will compute (s0|~s1)&s2	Syntax: TERNARY_REGIMM_REG_REGIMM
10110001b	Boolean Function 0xB1	This symbol will compute the following functions: s0&s2|~s0&~s1 (s0|s1)^(~s0|~s2) (s0|~s1)^s0&~s2 (s0|s1^s2)^~s2 (~s0|s2)^~s0&s1 (~s0|s1^s2)^s1 s0&s2^~s0&~s1 s0&(s1^~s2)^~s1 ~s0&(s1^~s2)^s2 (s0|~s1)&(~s0|s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
10110010b	Boolean Function 0xB2	This symbol will compute the following functions: s0&s2|(s0|s2)&~s1 s0&s2|(s0^s2)&~s1 s0&~s1|(s0|~s1)&s2 s0&~s1|(s0^~s1)&s2 s0&(~s1|s2)|~s1&s2 s0&(s1^~s2)|~s1&s2 s0^(s0^s2)&(s1^s2) s0^(s0^~s1)&(s1^s2) (s0|s2)^(s0^s2)&s1 (s0|~s1)^(s0^~s1)&~s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		(s0|s1^s2)^s1&~s2 (~s0|s1)^(s0^s1|~s2) (~s0|~s2)^(s0^~s2|s1) (~s0|s1^s2)^(s1|~s2) (s0^s1|s1^s2)^s1 (s0^s2|s1^s2)^s1 (s0^~s1|s1^s2)^~s2 (s0^~s2|s1^s2)^~s2 s0&s2^(s0^s2)&~s1 s0&~s1^(s0^~s1)&s2 s0&(s1^~s2)^~s1&s2 ~s0&s1^(s0^s1|s2) ~s0&~s2^(s0^~s2|~s1) ~s0&(s1^~s2)^(~s1|s2) (s0^s1)&(s1^~s2)^s2 (s0^s2)&(s1^~s2)^s2 (s0^~s1)&(s1^~s2)^~s1 (s0^~s2)&(s1^~s2)^~s1 (s0|s2)&(s0^~s2|~s1) (s0|s2)&(s0&s2|~s1) (s0|~s1)&(s0^s1|s2) (s0|~s1)&(s0&~s1|s2) (s0|s1^s2)&(~s1|s2) (s0|~s1&s2)&(~s1|s2)	
10110011b	Boolean Function 0xB3	This symbol will compute s0&s2|~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
10110100b	Boolean Function 0xB4	This symbol will compute the following functions: (s0|~s1)^~s2 ~s0&s1^s2	Syntax: TERNARY_REGIMM_REG_REGIMM
10110101b	Boolean Function 0xB5	This symbol will compute the following functions: s0^~s2|~s0&~s1 s0^~s2|~s1&s2 s0^(~s0&~s1|~s2) (s0|~s1&s2)^~s2 ~s0&(s1|~s2)^s2	Syntax: TERNARY_REGIMM_REG_REGIMM
10110110b	Boolean Function 0xB6	This symbol will compute the following functions: s0^s1&~s2|~s1&s2 s0^s1^s2|~s1&s2 s0^~s1^~s2|~s1&s2 s0&s2|(s0|s2)^s1 s0&s2|s0^s1^s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl			
		s0&s2|s0^~s1^~s2 s0&s2|~s0&~s2^~s1 s0&~s1|(s0|~s1)^~s2 s0&~s1|s0^s1^s2 s0&~s1|s0^~s1^~s2 s0&~s1|~s0&s1^s2 s0^(~s0|s1)&(s1^s2) s0^(~s0|~s2)&(s1^s2) (s0|s2)^(~s0|~s2)&s1 (s0|~s1)^(~s0|s1)&~s2 (s0|~s1&s2)^s1&~s2 (s0^s2|s0&~s1)^s1 (s0^s2|~s1&s2)^s1 (s0^~s1|s0&s2)^~s2 (s0^~s1|~s1&s2)^~s2 s0^(s0&s2|s1)^s2 s0^(s0&~s1|~s2)^~s1 s0^(~s0|s1)&s2^s1 s0^(~s0|~s2)&~s1^~s2 (~s0|~s1&s2)^s1^~s2 (~s0|~s1&s2)^~s1^s2 s0&(s1|~s2)^s1^s2 s0&(s1|~s2)^~s1^~s2 ~s0&s1^(s0&~s1|s2) ~s0&~s2^(s0&s2|~s1) ~s0&(s1|~s2)^(~s1|s2) (s0^s1)&(s1|~s2)^s2 (s0^s1)&(~s0|~s2)^s2 (s0^~s2)&(s1|~s2)^~s1 (s0^~s2)&(~s0|s1)^~s1	
10110111b	Boolean Function 0xB7	This symbol will compute s0^~s2|~s1	Syntax: TERNARY_REGIMM_REG_REGIMM
10111000b	Boolean Function 0xB8	This symbol will compute the following functions: s0&s1|~s1&s2 s0^(s0^s2)&~s1 (s0|~s1)^~s1&~s2 (~s0|~s1)^(s1|~s2) (s0^~s2|~s1)^~s2 s0&s1^~s1&s2 ~s0&s1^(s1|s2) (s0^s2)&s1^s2 (s0|~s1)&(s1|s2)	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl			

10111001b	Boolean Function 0xB9	This symbol will compute the following functions:
		`s0^~s1|s0&s2`
		`s0^~s1|~s1&s2`
		`s0^(~s0|~s2)&~s1`
		`(~s0|~s1&s2)^s1`
		`s0&(s1|~s2)^~s1`
			Syntax: TERNARY_REGIMM_REG_REGIMM

10111010b	Boolean Function 0xBA	This symbol will compute `s0|~s1&s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
10111011b	Boolean Function 0xBB	This symbol will compute `s0|~s1`	Syntax: TERNARY_REGIMM_REG_REGIMM

10111100b	Boolean Function 0xBC	This symbol will compute the following functions:
		`s0&s1|s1^s2`
		`s0&s2|s1^s2`
		`(s0&s1|~s2)^~s1`
		`(s0&s2|~s1)^~s2`
		`(~s0|~s1)&s2^s1`
		`(~s0|~s2)&s1^s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

10111101b	Boolean Function 0xBD	This symbol will compute the following functions:
		`s0^~s1|s1^s2`
		`s0^~s2|s1^s2`
			Syntax: TERNARY_REGIMM_REG_REGIMM

10111110b	Boolean Function 0xBE	This symbol will compute `s0|s1^s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
10111111b	Boolean Function 0xBF	This symbol will compute `s0|~s1|~s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11000000b	Boolean Function 0xC0	This symbol will compute `s1&s2`	Syntax: TERNARY_REGIMM_REG_REGIMM

11000001b	Boolean Function 0xC1	This symbol will compute the following functions:
		`(s0&~s1|s2)^~s1`
		`(s0&~s2|s1)^~s2`
		`(~s0|s1)&~s2^s1`
		`(~s0|s2)&~s1^s2`
		`(~s0|s1)&(s1^~s2)`
		`(~s0|s2)&(s1^~s2)`
			Syntax: TERNARY_REGIMM_REG_REGIMM

11000010b	Boolean Function 0xC2	This symbol will compute the following functions:
		`(~s0&~s1|s2)^~s1`
		`(~s0&~s2|s1)^~s2`
		`(s0|s1)&~s2^s1`
			Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		(s0|s2)&~s1^s2 (s0|s1)&(s1^~s2) (s0|s2)&(s1^~s2)	
11000011b	Boolean Function 0xC3	This symbol will compute s1^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11000100b	Boolean Function 0xC4	This symbol will compute (~s0|s2)&s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11000101b	Boolean Function 0xC5	This symbol will compute the following functions: ~s0&~s2|s1&s2 s0^(s0^s1|~s2) (s0|s2)^(~s1|~s2) (~s0|s2)^~s1&s2 (s0^s1|s2)^~s1 s0&~s2^(s1|~s2) ~s0&~s2^s1&s2 (s0^~s1)&~s2^s1 (~s0|s2)&(s1|~s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
11000110b	Boolean Function 0xC6	This symbol will compute the following functions: (~s0|s2)^~s1 s0&~s2^s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11000111b	Boolean Function 0xC7	This symbol will compute the following functions: ~s0&s1|s1^~s2 ~s0&~s2|s1^~s2 (~s0&s1|s2)^~s1 (~s0&~s2|~s1)^s2 (s0|s2)&s1^~s2 (s0|~s1)&~s2^s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11001000b	Boolean Function 0xC8	This symbol will compute (s0|s2)&s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11001001b	Boolean Function 0xC9	This symbol will compute the following functions: (s0|s2)^~s1 ~s0&~s2^s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11001010b	Boolean Function 0xCA	This symbol will compute the following functions: s0&~s2|s1&s2 s0^(s0^s1)&s2 (s0|s2)^~s1&s2 (~s0|s2)^(~s1|~s2)	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		(s0^~s1|s2)^~s1 s0&~s2^s1&s2 ~s0&~s2^(s1|~s2) (s0^s1)&~s2^s1 (s0|s2)&(s1|~s2)	
11001011b	Boolean Function 0xCB	This symbol will compute the following functions: s0&s1|s1^~s2 s0&~s2|s1^~s2 (s0&s1|s2)^~s1 (s0&~s2|~s1)^s2 (~s0|s2)&s1^~s2 (~s0|~s1)&~s2^s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11001100b	Boolean Function 0xCC	This symbol will compute s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11001101b	Boolean Function 0xCD	This symbol will compute ~s0&~s2|s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11001110b	Boolean Function 0xCE	This symbol will compute s0&~s2|s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11001111b	Boolean Function 0xCF	This symbol will compute s1|~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11010000b	Boolean Function 0xD0	This symbol will compute (~s0|s1)&s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11010001b	Boolean Function 0xD1	This symbol will compute the following functions: ~s0&~s1|s1&s2 s0^(s0^s2|~s1) (s0|s1)^(~s1|~s2) (~s0|s1)^s1&~s2 (s0^s2|s1)^~s2 s0&~s1^(~s1|s2) ~s0&~s1^s1&s2 (s0^~s2)&~s1^s2 (~s0|s1)&(~s1|s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
11010010b	Boolean Function 0xD2	This symbol will compute the following functions: (~s0|s1)^~s2 s0&~s1^s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11010011b	Boolean Function 0xD3	This symbol will compute the following functions: ~s0&s2|s1^~s2 ~s0&~s1|s1^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		(~s0&s2|s1)^~s2 (~s0&~s1|~s2)^s1 (s0|s1)&s2^~s1 (s0|~s2)&~s1^s2	
11010100b	Boolean Function 0xD4	This symbol will compute the following functions: ~s0&s1|(~s0|s1)&s2 ~s0&s1|(s0^~s1)&s2 ~s0&s2|(~s0|s2)&s1 ~s0&s2|(s0^~s2)&s1 ~s0&(s1|s2)|s1&s2 ~s0&(s1^s2)|s1&s2 s0^(s0^s1|s1^s2) s0^(s0^s2|s1^s2) (s0|~s1)^(s0^s1|~s2) (s0|~s2)^(s0^s2|~s1) (s0|s1^~s2)^(~s1|~s2) (~s0|s1)^(s0^~s1)&~s2 (~s0|s2)^(s0^~s2)&~s1 (~s0|s1^~s2)^~s1&~s2 (s0^s1|s1^~s2)^~s1 (s0^s2|s1^~s2)^~s2 (s0^~s1|s1^~s2)^~s2 (s0^~s2|s1^~s2)^~s1 s0&~s1^(s0^s1|s2) s0&~s2^(s0^s2|s1) s0&(s1^s2)^(s1|s2) ~s0&s1^(s0^~s1)&s2 ~s0&s2^(s0^~s2)&s1 ~s0&(s1^s2)^s1&s2 (s0^s1)&(s1^s2)^s2 (s0^s2)&(s1^s2)^s1 (s0^~s1)&(s1^s2)^s1 (s0^~s2)&(s1^s2)^s2 (~s0|s1)&(s0^s1|s2) (~s0|s1)&(~s0&s1|s2) (~s0|s2)&(s0^s2|s1) (~s0|s2)&(~s0&s2|s1) (~s0|s1^~s2)&(s1|s2) (~s0|s1&s2)&(s1|s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
11010101b	Boolean Function 0xD5	This symbol will compute ~s0|s1&s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11010110b	Boolean Function 0xD6	This symbol will compute the following functions: s0^(s1|s2)|s1&s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		s0^s1^s2|s1&s2 s0^~s1^~s2|s1&s2 ~s0&s1|(~s0|s1)^~s2 ~s0&s1|s0^s1^s2 ~s0&s1|s0^~s1^~s2 ~s0&s1|s0&~s1^s2 ~s0&s2|(~s0|s2)^~s1 ~s0&s2|s0^s1^s2 ~s0&s2|s0^~s1^~s2 ~s0&s2|s0&~s2^s1 s0^(~s0&s1|s1^s2) s0^(~s0&s2|s1^s2) (~s0|s1)^(s0|~s1)&~s2 (~s0|s2)^(s0|~s2)&~s1 (~s0|s1&s2)^~s1&~s2 (s0^~s1|s1&s2)^~s2 (s0^~s1|~s0&s2)^~s2 (s0^~s2|s1&s2)^~s1 (s0^~s2|~s0&s1)^~s1 s0^(~s0&s1|~s2)^~s1 s0^(~s0&s2|~s1)^~s2 s0^(s0|~s1)&s2^s1 s0^(s0|~s2)&s1^s2 (s0|s1&s2)^s1^s2 (s0|s1&s2)^~s1^~s2 ~s0&(~s1|~s2)^s1^~s2 ~s0&(~s1|~s2)^~s1^s2 s0&~s1^(~s0&s1|s2) s0&~s2^(~s0&s2|s1) s0&(~s1|~s2)^(s1|s2) (s0|~s1)&(s0^s2)^s1 (s0|~s2)&(s0^s1)^s2 (s0^s1)&(~s1|~s2)^s2 (s0^s2)&(~s1|~s2)^s1	
11010111b	Boolean Function 0xD7	This symbol will compute ~s0|s1^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11011000b	Boolean Function 0xD8	This symbol will compute the following functions: s0&s1|~s0&s2 (s0|s2)^s0&~s1 (s0|~s2)^(~s0|~s1) (s0|s1^~s2)^~s1 (~s0|s1)^~s0&~s2 (~s0|s1^~s2)^~s2 s0&s1^~s0&s2 s0&(s1^s2)^s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		~s0&(s1^s2)^s1 (s0|s2)&(~s0|s1)	
11011001b	Boolean Function 0xD9	This symbol will compute the following functions: s0^~s1|s1&s2 s0^~s1|~s0&s2 s0^(~s0&s2|~s1) (s0|s1&s2)^~s1 ~s0&(~s1|~s2)^s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11011010b	Boolean Function 0xDA	This symbol will compute the following functions: s0^s2|s0&s1 s0^s2|s1&s2 s0^(~s0|~s1)&s2 (~s0|s1&s2)^~s2 s0&(~s1|~s2)^s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11011011b	Boolean Function 0xDB	This symbol will compute the following functions: s0^s2|s1^~s2 s0^~s1|s1^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11011100b	Boolean Function 0xDC	This symbol will compute ~s0&s2|s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11011101b	Boolean Function 0xDD	This symbol will compute ~s0|s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11011110b	Boolean Function 0xDE	This symbol will compute s0^s2|s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11011111b	Boolean Function 0xDF	This symbol will compute ~s0|s1|~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11100000b	Boolean Function 0xE0	This symbol will compute (s0|s1)&s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11100001b	Boolean Function 0xE1	This symbol will compute the following functions: (s0|s1)^~s2 ~s0&~s1^s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11100010b	Boolean Function 0xE2	This symbol will compute the following functions: s0&~s1|s1&s2 s0^(s0^s2)&s1 (s0|s1)^s1&~s2 (~s0|s1)^(~s1|~s2) (s0^~s2|s1)^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl			
		s0&~s1^s1&s2 ~s0&~s1^(~s1|s2) (s0^s2)&~s1^s2 (s0|s1)&(~s1|s2)	
11100011b	Boolean Function 0xE3	This symbol will compute the following functions: s0&s2|s1^~s2 s0&~s1|s1^~s2 (s0&s2|s1)^~s2 (s0&~s1|~s2)^s1 (~s0|s1)&s2^~s1 (~s0|~s2)&~s1^s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11100100b	Boolean Function 0xE4	This symbol will compute the following functions: s0&s2|~s0&s1 (s0|s1)^s0&~s2 (s0|~s1)^(~s0|~s2) (s0|s1^~s2)^~s2 (~s0|s2)^~s0&~s1 (~s0|s1^~s2)^~s1 s0&s2^~s0&s1 s0&(s1^s2)^s1 ~s0&(s1^s2)^s2 (s0|s1)&(~s0|s2)	Syntax: TERNARY_REGIMM_REG_REGIMM
11100101b	Boolean Function 0xE5	This symbol will compute the following functions: s0^~s2|s1&s2 s0^~s2|~s0&s1 s0^(~s0&s1|~s2) (s0|s1&s2)^~s2 ~s0&(~s1|~s2)^s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11100110b	Boolean Function 0xE6	This symbol will compute the following functions: s0^s1|s0&s2 s0^s1|s1&s2 s0^(~s0|~s2)&s1 (~s0|s1&s2)^~s1 s0&(~s1|~s2)^s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11100111b	Boolean Function 0xE7	This symbol will compute the following functions: s0^s1|s1^~s2 s0^~s2|s1^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11101000b	Boolean Function 0xE8	This symbol will compute the following functions:	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		s0&s1|(s0|s1)&s2 s0&s1|(s0^s1)&s2 s0&s2|(s0|s2)&s1 s0&s2|(s0^s2)&s1 s0&(s1|s2)|s1&s2 s0&(s1^s2)|s1&s2 s0^(s0^s1)&(s1^~s2) s0^(s0^s2)&(s1^~s2) (s0|s1)^(s0^s1)&~s2 (s0|s2)^(s0^s2)&~s1 (s0|s1^~s2)^~s1&~s2 (~s0|~s1)^(s0^~s1|~s2) (~s0|~s2)^(s0^~s2|~s1) (~s0|s1^~s2)^(~s1|~s2) (s0^s1|s1^~s2)^~s2 (s0^s2|s1^~s2)^~s1 (s0^~s1|s1^~s2)^~s1 (s0^~s2|s1^~s2)^~s2 s0&s1^(s0^s1)&s2 s0&s2^(s0^s2)&s1 s0&(s1^s2)^s1&s2 ~s0&~s1^(s0^~s1|s2) ~s0&~s2^(s0^~s2|s1) ~s0&(s1^s2)^(s1|s2) (s0^s1)&(s1^s2)^s1 (s0^s2)&(s1^s2)^s2 (s0^~s1)&(s1^s2)^s2 (s0^~s2)&(s1^s2)^s1 (s0|s1)&(s0^~s1|s2) (s0|s1)&(s0&s1|s2) (s0|s2)&(s0^~s2|s1) (s0|s2)&(s0&s2|s1) (s0|s1^~s2)&(s1|s2) (s0|s1&s2)&(s1|s2)	
11101001b	Boolean Function 0xE9	This symbol will compute the following functions: s0^~s1&~s2|s1&s2 s0^s1^~s2|s1&s2 s0^~s1^s2|s1&s2 s0&s1|(s0|s1)^~s2 s0&s1|s0^s1^~s2 s0&s1|s0^~s1^s2 s0&s1|~s0&~s1^s2 s0&s2|(s0|s2)^~s1 s0&s2|s0^s1^~s2 s0&s2|s0^~s1^s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl

		s0&s2|~s0&~s2^s1 s0^(~s0|~s1)&(s1^~s2) s0^(~s0|~s2)&(s1^~s2) (s0|s1)^(~s0|~s1)&~s2 (s0|s2)^(~s0|~s2)&~s1 (s0|s1&s2)^~s1&~s2 (s0^s1|s0&s2)^~s2 (s0^s1|s1&s2)^~s2 (s0^s2|s0&s1)^~s1 (s0^s2|s1&s2)^~s1 s0^(s0&s1|~s2)^s1 s0^(s0&s2|~s1)^s2 s0^(~s0|~s1)&s2^~s1 s0^(~s0|~s2)&s1^~s2 (~s0|s1&s2)^s1^s2 (~s0|s1&s2)^~s1^~s2 s0&(~s1|~s2)^s1^~s2 s0&(~s1|~s2)^~s1^s2 ~s0&~s1^(s0&s1|s2) ~s0&~s2^(s0&s2|s1) ~s0&(~s1|~s2)^(s1|s2) (s0^~s1)&(~s0|~s2)^s2 (s0^~s1)&(~s1|~s2)^s2 (s0^~s2)&(~s0|~s1)^s1 (s0^~s2)&(~s1|~s2)^s1	
11101010b	Boolean Function 0xEA	This symbol will compute s0|s1&s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11101011b	Boolean Function 0xEB	This symbol will compute s0|s1^~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11101100b	Boolean Function 0xEC	This symbol will compute s0&s2|s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11101101b	Boolean Function 0xED	This symbol will compute s0^~s2|s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11101110b	Boolean Function 0xEE	This symbol will compute s0|s1	Syntax: TERNARY_REGIMM_REG_REGIMM
11101111b	Boolean Function 0xEF	This symbol will compute s0|s1|~s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11110000b	Boolean Function 0xF0	This symbol will compute s2	Syntax: TERNARY_REGIMM_REG_REGIMM
11110001b	Boolean Function 0xF1	This symbol will compute ~s0&~s1|s2	Syntax: TERNARY_REGIMM_REG_REGIMM

BooleanFuncCtrl			
11110010b	Boolean Function 0xF2	This symbol will compute `s0&~s1|s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11110011b	Boolean Function 0xF3	This symbol will compute `~s1|s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11110100b	Boolean Function 0xF4	This symbol will compute `~s0&s1|s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11110101b	Boolean Function 0xF5	This symbol will compute `~s0|s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11110110b	Boolean Function 0xF6	This symbol will compute `s0^s1|s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11110111b	Boolean Function 0xF7	This symbol will compute `~s0|~s1|s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11111000b	Boolean Function 0xF8	This symbol will compute `s0&s1|s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11111001b	Boolean Function 0xF9	This symbol will compute `s0^~s1|s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11111010b	Boolean Function 0xFA	This symbol will compute `s0|s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11111011b	Boolean Function 0xFB	This symbol will compute `s0|~s1|s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11111100b	Boolean Function 0xFC	This symbol will compute `s1|s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11111101b	Boolean Function 0xFD	This symbol will compute `~s0|s1|s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11111110b	Boolean Function 0xFE	This symbol will compute `s0|s1|s2`	Syntax: TERNARY_REGIMM_REG_REGIMM
11111111b	Boolean Function 0xFF	This symbol will compute `1` (the one's function)	Syntax: TERNARY_REGIMM_REG_REGIMM

ChanOff

ChanOff	
Source:	EuIsa
Size (in bits):	3

Channel Offset

This enumeration (instruction 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. Some of the ARFs affected by this offset are:

- ce access (channel enable / the execution mask): bitwise offset
- f# access: bitwise offset (e.g. in predication or conditional modifier access)
- acc# access via AccWrEn (implicit only): in subregister units of the data type

Note: ChanOff is functionally the same as a concatenation of the previous QtrCtrl and NibCtrl and supersedes these fields.

Restriction
The execution size (ExecSize) must be a factor of the chosen offset. For instance, M28 (offset 28) can only be used with SIMD1, SIMD2, and SIMD4.

Value	Name	Description
000b	M0 **[Default]**	ARF access begins at channel 0
001b	M4	ARF access begins at channel 4 (e.g. 4 bits in on ce and f#
010b	M8	
011b	M12	
100b	M16	
101b	M20	
110b	M24	
111b	M28	

COMPONENT_ENABLES

COMPONENT_ENABLES	
Source:	RenderCS
Size (in bits):	4

If enabled, the component will be stored in the URB.

Value	Name
0000b	NONE
0001b	X
0010b	Y
0011b	XY
0100b	Z
0101b	XZ
0110b	YZ
0111b	XYZ
1000b	W
1001b	XW
1010b	YW
1011b	XYW
1100b	ZW
1101b	XZW
1110b	YZW
1111b	XYZW

DP_ADDR_REG_SIZE

DP_ADDR_REG_SIZE	
Source:	SFID_1, SFID_D, SFID_E, SFID_F
Size (in bits):	4

Specifies the size of the dataport address payload in registers.

Programming Notes	Source
Untyped memory address payloads are 1 phase long.	SFID_1, SFID_E, SFID_F
Typed global memory address payloads are 1-4 phases long, selected by the surface type.	SFID_D

Value	Name	Programming Notes
1		A16 buffer payload A16 1D payload (U)
2		A32 buffer payload A32 1D payload (U) A16 2D payload (U, V)
3		A16 3D payload (U, V, R)
4		A64 buffer payload A32 2D payload (U, V) A16 3D payload (U, V, R, LOD)
6		A32 3D payload (U, V, R)
8		A32 3D payload (U, V, R, LOD)

DP_ADDR_SIZE

DP_ADDR_SIZE	
Source:	SFID_1, SFID_6, SFID_D, SFID_E, SFID_F
Size (in bits):	2

Specifies the size of the address payload item in a dataport message.

Programming Notes
One 256-bit register holds 16 A16 values, or 8 A32 values, or 4 A64 values.

Restriction
Dataports UGM, UGML and TGM do not support addr_size A16.

Value	Name	Description	Programming Notes
2	A32	32-bit address offset	Register format is A32_PAYLOAD_SIMT16 for SIMT16 messages, or A32_PAYLOAD_SIMT8 for SIMT8 messages. Data port TGM (SFID_D) messages are limited to SIMT16. Register format is A32_PAYLOAD_SIMT16.
3	A64	64-bit address offset	Register format is A64_PAYLOAD_SIMT16 for SIMT16 messages, or A64_PAYLOAD_SIMT8 for SIMT8 messages. A64 is only allowed when DP_ADDR_SURFACE_TYPE in the message descriptor is programmed as FLAT.

DP_ADDR_SURFACE_TYPE

DP_ADDR_SURFACE_TYPE	
Source:	SFID_1, SFID_D, SFID_E, SFID_F
Size (in bits):	2

Specifies the type of the address payload item in a dataport message. The address type specifies how the dataport message decodes the Extended Descriptor for the surface attributes and address calculation.

Programming Notes
The 64-bit Base address is specified by the selected surface. For stateless accesses, The BaseOffset from ExtDesc is a signed offset from the Base address.

Restriction
SLM dataport messages do not support surfaces. TGM dataport messages do not support stateless accesses.
A64 address size must use Address type FLAT.
UGML dataport (SFID_1) does not support addr_type FLAT with addr_size A32 or A16. UGML supports addr_type FLAT with A64 only.

Value	Name	Description
0	Flat	Extended Descriptor format is **EXDESC_FLAT**.
3	Binding Table Index	Extended Descriptor format is **EXDESC_BTI**.
1	Bindless Surface State	Extended Descriptor format is **EXDESC_SURFACE**.
2	Surface State	Extended Descriptor format is **EXDESC_SURFACE**.

DP_CACHE_LOAD

DP_CACHE_LOAD	
Source:	SFID_1, SFID_6, SFID_D, SFID_E, SFID_F
Size (in bits):	3

Specifies the dataport message override to the default L1 and L3 memory cache policies. Dataport L1 cache policies are uncached (UC), cached (C), cache streaming (S) and invalidate-after-read (IAR). Dataport L3 cache policies are uncached (UC) and cached (C).

Programming Notes
This message override always maps to a specific cache policy. However, some dataport messages do not support some cache policies. Unsupported policies are re-mapped by hardware to one of the message's supported settings.

Value	Name	Description	Programming Notes
0	L1STATE_L3MOCS **[Default]**	No override. Use the non-pipelined state or surface state cache settings for L1 and L3	
1	L1UC_L3UC	Override to L1 uncached and L3 uncached	
2	L1UC_L3C	Override to L1 uncached and L3 cached	
3	L1C_L3UC	Override to L1 cached and L3 uncached	
4	L1C_L3C	Override to cache at both L1 and L3	
5	L1S_L3UC	Override to L1 streaming load and L3 uncached	
6	L1S_L3C	Override to L1 streaming load and L3 cached	
7	L1IAR_L3C	For load messages, override to L1 invalidate-after-read, and L3 cached.	This value is not allowed for 'prefetch' messages.

DP_CACHE_STORE

DP_CACHE_STORE	
Source:	SFID_1, SFID_6, SFID_D, SFID_F
Size (in bits):	3

Specifies the dataport message override to the default L1 and L3 memory cache policies. Dataport L1 cache policies are uncached (UC), write-through (WT), write-back (WB) and streaming (S). Dataport L3 cache policies are uncached (UC) and cached (WB).

Programming Notes

This message override always maps to a specific cache policy. However, some dataport messages do not support some cache policies. Unsupported policies are re-mapped by hardware to one of the message's supported settings.

Value	Name	Description
0	L1STATE_L3MOCS **[Default]**	No override. Use the non-pipelined or surface state cache settings for L1 and L3
1	L1UC_L3UC	Override to L1 uncached and L3 uncached
2	L1UC_L3WB	Override to L1 uncached and L3 cached
3	L1WT_L3UC	Override to L1 write-through and L3 uncached
4	L1WT_L3WB	Override to L1 write-through and L3 cached
5	L1S_L3UC	Override to L1 streaming and L3 uncached
6	L1S_L3WB	Override to L1 streaming and L3 cached
7	L1WB_L3WB	Override to L1 write-back, and L3 cached

DP_CMASK

DP_CMASK	
Source:	SFID_1, SFID_D, SFID_E, SFID_F
Size (in bits):	4

Specifies which components of the data payload 4-element vector (X, Y, Z, W) is packed into the register payload.

Programming Notes
Transpose is not supported on CMASK messages.

Restriction	Source
Restriction : Store messages to SFID_1 (UGML) only supports contiguous vectors (X=V1, XY=V2, XYZ=V3, XYZW=V4).	SFID_1

Value	Name
0x1	X
0x2	Y
0x4	Z
0x8	W
0x3	XY
0x5	XZ
0x9	XW
0x6	YZ
0xA	YW
0xC	ZW
0x7	XYZ
0xB	XYW
0xD	XZW
0xE	YZW
0xF	XYZW **[Default]**

DP_DATA_SIZE

DP_DATA_SIZE	
Source:	SFID_1, SFID_6, SFID_D, SFID_E, SFID_F
Size (in bits):	3

Specifies the size of the data payload item in a dataport message.

Programming Notes
One 256-bit register holds 32 D8 values, or 16 D16 values, or 8 D32/D8U32/D16U32 values, or 4 D64 values.

Restriction	Source
D8 and D16 are not supported, except for the 2D Block load/store messages. Use D8U32 and D16U32. D8U32 and D16U32 are not supported in the 2D Block load/store messages.	
Restriction : D8 and D8U32 are not supported for any atomic operations.	SFID_1, SFID_D, SFID_E, SFID_F
Restriction : This field is ignored for typed messages. Typed global memory load/store messages pack each component into a 32-bit value in register. The data size in memory is obtained from data format specified in the surface state.	SFID_D

Value	Name	Description	Programming Notes	Source
0	D8	8-bit scalar data value in memory, packed into a 8-bit data value in register	Register format D8_PAYLOAD	SFID_1, SFID_E, SFID_F
1	D16	16-bit scalar data value in memory, packed into a 16-bit data value in register	Register format D16_PAYLOAD	SFID_1, SFID_E, SFID_F
2	D32	32-bit scalar data value in memory, packed into 32-bit data value in register	For SIMT16 messages: Register format is D32_2SRC_ATM_PAYLOAD_SIMT16 for 2-source atomic messages, and D32_DATA_PAYLOAD_SIMT16 for all other messages. For SIMT8 messages: Register format is D32_2SRC_ATM_PAYLOAD_SIMT8 for 2-source atomic messages, and D32_DATA_PAYLOAD_SIMT8 for all other messages.	SFID_1, SFID_6, SFID_D, SFID_E, SFID_F
3	D64	64-bit scalar data value in memory, packed into 64-bit data value in register	For SIMT16 messages: Register format is D64_2SRC_ATM_PAYLOAD_SIMT16 for 2-source atomic messages, and D64_DATA_PAYLOAD_SIMT16 for all other messages. For SIMT8 messages: Register format is D64_2SRC_ATM_PAYLOAD_SIMT8for 2-source atomic messages, and D64_DATA_PAYLOAD_SIMT8for all other messages.	SFID_1, SFID_E, SFID_F

DP_DATA_SIZE				
4	D8U32	8-bit scalar data value in memory, packed into 32-bit unsigned data value in register	For SIMT16 messages: Register format is D32_2SRC_ATM_PAYLOAD_SIMT16 for 2-source atomic messages, and D32_DATA_PAYLOAD_SIMT16 for all other messages. For SIMT8 messages: Register format is D32_2SRC_ATM_PAYLOAD_SIMT8 for 2-source atomic messages, and D32_DATA_PAYLOAD_SIMT8 for all other messages.	SFID_1, SFID_D, SFID_E, SFID_F
5	D16U32	16-bit scalar data value in memory, packed into 32-bit unsigned data value in register	For SIMT16 messages: Register format is D32_2SRC_ATM_PAYLOAD_SIMT16 for 2-source atomic messages, and D32_DATA_PAYLOAD_SIMT16 for all other messages. For SIMT8 messages: Register format is D32_2SRC_ATM_PAYLOAD_SIMT8 for 2-source atomic messages, and D32_DATA_PAYLOAD_SIMT8 for all other messages.	SFID_1, SFID_6, SFID_D, SFID_E, SFID_F

DP_FENCE_SCOPE

DP_FENCE_SCOPE		
Size (in bits): 3		
Specifies the scope of the fence.		
Programming Notes		
For device-scope fence on a single-tile GPU, set the scope to "Tile" for best performance.		
Value	**Name**	**Description**
0	Threadgroup **[Default]**	Wait until all previous memory transactions from this thread are observed within the local thread-group.
1	Local	Wait until all previous memory transactions from this thread are observed within the local sub-slice.
2	Tile	Wait until all previous memory transactions from this thread are observed in the local tile.
3	GPU	Wait until all previous memory transactions from this thread are observed in the local GPU.
4	All GPU	Wait until all previous memory transactions from this thread are observed across all GPUs in the system.

DP_FLUSH_TYPE

DP_FLUSH_TYPE		
Size (in bits): 3		
Specifies the type of cache flush operation to perform after a fence is complete.		
Programming Notes		
After fence completes, any flush operation is applied sequentially from the narrowest scope out to this scope level.		
Value	**Name**	**Description**
6	None	
1	Evict	For a R/W cache, evict dirty lines (M to I state) and invalidate clean lines. For a RO cache, invalidate clean lines.
2	Invalidate	For both R/W and RO cache, invalidate clean lines in the cache.
3	Discard	For a R/W cache, invalidate dirty lines (M to I state), without write-back to next level. This opcode does nothing for a RO cache.
4	Clean	For a R/W cache, write-back dirty lines to the next level, but kept in the cache as "clean" (M to V state). This opcode does nothing for a RO cache.
5	DGT_L3Only Flush	Flush "RW" section of the L3 cache, but leave L1 and L2 caches untouched.

DP_ONE_ADDR_REG

DP_ONE_ADDR_REG			
Source:	SFID_1, SFID_D, SFID_E, SFID_F		
Size (in bits):	1		
Specifies the size of the dataport address payload in registers.			
Value	**Name**	**Programming Notes**	**Source**
1		ABLOCK_PAYLOAD for LOAD_BLOCK/STORE_BLOCK messages A2DBLOCK_PAYLOAD for LOAD_2DBLOCK/STORE_2DBLOCK messages ASTATE_INFO_PAYLOAD for RSI message. ADDR_FENCE_PAYLOAD for UGM, UGML, TGM and SLM fence messages.	SFID_1, SFID_D, SFID_E, SFID_F

DP_OPCODE

DP_OPCODE	
Source:	SFID_1, SFID_D, SFID_F
Size (in bits):	6

Specifies the dataport send instruction opcode

Value	Name	Description
0	LOAD	Load operation
1	LOAD_BLOCK	Load_block operation
2	LOAD_CMASK	Load cmask
3	LOAD_2DBLOCK	LOAD_2DBLOCK
4	STORE	STORE
5	STORE_BLOCK	STORE_BLOCK
6	STORE_CMASK	STORE_CMASK
7	STORE_2DBLOCK	STORE_2DBLOCK
8	ATOMIC_INC	ATOMIC_INC
9	ATOMIC_DEC	ATOMIC_DEC
10	ATOMIC_LOAD	ATOMIC_LOAD
11	ATOMIC_STORE	ATOMIC_STORE
12	ATOMIC_ADD	ATOMIC_ADD
13	ATOMIC_SUB	ATOMIC_SUB
14	ATOMIC_MIN	ATOMIC_MIN
15	ATOMIC_MAX	ATOMIC_MAX
16	ATOMIC_UMIN	ATOMIC_UMIN
17	ATOMIC_UMAX	ATOMIC_UMAX
18	ATOMIC_CMPXCHG	ATOMIC_CMPXCHG
19	ATOMIC_FADD	ATOMIC_FADD
20	ATOMIC_FSUB	ATOMIC_FSUB
21	ATOMIC_FMIN	ATOMIC_FMIN
22	ATOMIC_FMAX	ATOMIC_FMAX
23	ATOMIC_FCMPXCHG	ATOMIC_FCMPXCHG
24	ATOMIC_AND	ATOMIC_AND
25	ATOMIC_OR	ATOMIC_OR
26	ATOMIC_XOR	ATOMIC_XOR
27	LOAD_WITH_STATUS	LOAD_WITH_STATUS
28	STORE_UNCOMPRESSED	STORE_UNCOMPRESSED
29	CCS_UPDATE	CCS_UPDATE

DP_OPCODE		
30	RSI	Read State Information
31	FENCE	FENCE
32	ATOMIC_BFADD	ATOMIC_BFADD
33	ATOMIC_BFSUB	ATOMIC_BFSUB
34	ATOMIC_BFMIN	ATOMIC_BFMIN
35	ATOMIC_BFMAX	ATOMIC_BFMAX

DP_TRANSPOSE

DP_TRANSPOSE	
Source:	SFID_1, SFID_D, SFID_E, SFID_F
Size (in bits):	1

Specifies if the data payload vector is packed into the register in the normal SIMT layout, or is packed transposed in the SIMD layout.

Restriction
Transposed vectors are restricted to Exec_Mask = 1.

Value	Name	Description
0	Default **[Default]**	Normal SIMT layout in register
1	Transpose	Transposed into SIMD layout in register

DP_VECT_SIZE

DP_VECT_SIZE	
Source:	SFID_1, SFID_D, SFID_E, SFID_F
Size (in bits):	3

Specifies the size of the vector in a dataport message.

Restriction
Maximum vector size is 8 registers. The data size limits the dataport message's vector size. For example, with 512-bit registers and transpose enabled (simd1), the maximum vector size is D16.V16 or D32.V8 or D64.V4

Value	Name	Description
0	V1 **[Default]**	Vector length 1
1	V2	Vector length 2
2	V3	Vector length 3
3	V4	Vector length 4
4	V8	Vector length 8
5	V16	Vector length 16
6	V32	Vector length 32
7	V64	Vector length 64

EU_OPCODE

EU_OPCODE	
Source:	EuIsa
Size (in bits):	7

Value	Name	Properties
40h	add	Instruction: add
52h	add3	Instruction: add3
4Eh	addc	Instruction: addc
65h	and	Instruction: and
6Ch	asr	Instruction: asr
42h	avg	Instruction: avg
78h	bfe	Instruction: bfe
79h	bfi1	Instruction: bfi1
7Ah	bfi2	Instruction: bfi2
77h	bfrev	Instruction: bfrev
23h	brc	Instruction: brc
21h	brd	Instruction: brd
28h	break	Instruction: break
2Ch	call	Instruction: call
2Bh	calla	Instruction: calla
4Dh	cbit	Instruction: cbit
70h	cmp	Instruction: cmp
71h	cmpn	Instruction: cmpn
29h	cont	Instruction: cont
72h	csel	Instruction: csel
59h	dpas	Instruction: dpas
5Ah	dpasw	Instruction: dpasw
24h	else	Instruction: else
25h	endif	Instruction: endif
4Bh	fbh	Instruction: fbh

EU_OPCODE		
4Ch	fbl	Instruction: fbl
43h	frc	Instruction: frc
2Eh	goto	Instruction: goto
2Ah	halt	Instruction: halt
22h	if	Instruction: if
0h	illegal	Instruction: illegal
20h	jmpi	Instruction: jmpi
2Fh	join	Instruction: join
6Bh	bfn	Instruction: bfn
4Ah	lzd	Instruction: lzd
48h	mac	Instruction: mac
49h	mach	Instruction: mach
5Bh	mad	Instruction: mad
5Dh	madm	Instruction: madm
38h	math	Instruction: math
61h	mov	Instruction: mov
63h	movi	Instruction: movi
41h	mul	Instruction: mul
60h	nop	Instruction: nop
64h	not	Instruction: not
66h	or	Instruction: or
2Dh	ret	Instruction: ret
45h	rndd	Instruction: rndd
46h	rnde	Instruction: rnde
44h	rndu	Instruction: rndu
47h	rndz	Instruction: rndz
6Fh	rol	Instruction: rol
6Eh	ror	Instruction: ror
62h	sel	Instruction: sel

EU_OPCODE		
31h	send	Instruction: send
32h	sendc	Instruction: sendc
69h	shl	Instruction: shl
68h	shr	Instruction: shr
4Fh	subb	Instruction: subb
1h	sync	Instruction: sync
27h	while	Instruction: while
67h	xor	Instruction: xor

ExecSize

ExecSize	
Source:	EuIsa
Size (in bits):	3

Execution Size 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.

Restriction
An operand's Width must be less-than-or-equal to ExecSize

Value	Name	Programming Notes
000b	1 Channel (Scalar operation) **[Default]**	
001b	2 Channels	
010b	4 Channels	
011b	8 Channels	
100b	16 Channels	4-byte or smaller data types. Excludes DF, Q, and UQ types.
101b	32 Channels	2-byte or 1-byte data types. Excludes D, DF, F, Q, UD, and UQ types.
110b-111b	Reserved	

Fixed Function ID

FFID - Fixed Function ID

Size (in bits): 4

Fixed functions are hardware units that execute complex graphics or media command on behalf of application software. Some fixed functions send work down a pipeline through a series of fixed functions. Multiple fixed functions can be running at the same time. The GPU tracks activity from the fixed function with its FFID.

Programming Notes

Software does not specify the FFID, and does not normally use the FFID value. The FFID value is available to an EU thread in an ARF.

Value	Name	Description
00h	Null	
01h	Task Shader	
02h	Mesh Shader	
04h	Hull Shader	
05h	Domain Shader	
06h	Texel Shader	Adaptive Multi-Frequency Shader
07h	General Purpose Thread Spawner	GPGPU command queue thread dispatcher
08h	General Purpose Asynchronous Thread Spawner	GPGPU command queue's thread dispatcher for secondary queue
09h	Vertex Shader	
0Ch	Geometry Shader	
0Ah	CCS Continuation shader	BTD thread dispatch from SPAWN message, from Compute CS context.
0Bh	CCS Trace Ray shader	BTD thread dispatch from TRACE_RAY message, from Compute CS context.
0Dh	RCS Continuation shader	BTD thread dispatch from SPAWN message, from Render CS context.
0Eh	RCS Trace Ray shader	BTD thread dispatch from TRACE_RAY message, from Render CS context.
0Fh	Pixel Shader	This FFID indicates Pixel Shader Dispatch.

FlagModifier

FlagModifier	
Source:	EuIsa
Size (in bits):	4

Flag (Conditional) Modifier - This field sets the flag register based on the internal conditional signals output from the execution pipe such as sign, zero, overflow and NaNs, etc. If this field is set to 0000, no flag registers are updated. Flag registers are not updated for instructions with embedded compares. This field may also be referred to as the flag destination control field.

Value	Name	Description
0000b	None **[Default]**	None
0001b	(ze)	Zero
0010b	(nz)	NotZero
0011b	(gt)	Greater-than
0100b	(ge)	Greater-than-or-equal
0101b	(lt)	Less-than
0110b	(le)	Less-than-or-equal
0111b	Reserved	
1000b	(ov)	Overflow
1001b	(un)	Unordered (NaN)
1110b-1111b	Reserved	

GW_FENCE_PORTS

GW_FENCE_PORTS	
Source:	BSpec
Size (in bits):	0

Bit mask specifies the list of data ports to be fenced.

Value	Name	Description
0	None **[Default]**	No fence is performed when no bits are set.

HorzStride

HorzStride	
Source:	EuIsa
Size (in bits):	2

Horizontal Stride 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. This field applies to both destination and source operands. This field is not present for an immediate source operand.

A horizontal stride of 0 is used for a row that is one-element wide, useful when an instruction repeats a column value or repeats a scalar value. For example, adding a single column to every column in a 2D array or adding a scalar to every element in a 2D array uses HorzStride of 0. A horizontal stride of 1 indicates that elements are adjacent within a row. References to HorzStride in this volume normally reference the value not the encoding, so there are references to HorzStride of 4, which is encoded as 11b.

Value	Name
00b	0 elements
01b	1 elements
10b	2 elements
11b	4 elements

ImmDataType

ImmDataType	
Source:	EuIsa
Size (in bits):	4

Numeric data type of source and destination operand. Three source instructions use a 3-bit encoding that allows fewer data types.

Value	Name	Description
0000b	:uv	Packed Unsigned Half-Byte Integer Vector, 8 x 4-Bit Unsigned Integer
0001b	:uw	Unsigned Word (16-bit) Integer
0010b	:ud	Unsigned DoubleWord (32-bit) Integer
0011b	:uq	Unsigned Quadword (64-bit) Integer
0100b	:v	Packed Signed Half-Byte Integer Vector, 8 x 4-Bit Signed Integer
0101b	:w	Signed Word (16-bit) Integer
0110b	:d	Signed DoubleWord (32-bit) Integer
0111b	:q	Signed QuadWord (64-bit) Integer
1000b	:vf	Packed Restricted Float Vector, 4 x 8-Bit Restricted Precision Floating-Point Number.
1001b	:hf	Half (16-bit) Float
1010b	:f	Single-Precision (32-bit) Float
1011b	:df	Double-Precision (64-bit) Float
1100b	Reserved	
1101b	Reserved	
[1110b-1111b]	Reserved	

MathFC

MathFC	
Source:	EuIsa
Size (in bits):	4

Math Function Control

Value	Name	Description	Programming Notes	
0001b	INV	Reciprocal (Multiplicative Inverse): 1/src0	Table:special value processing | Src | +inf | +0 / +Denorm | -0 / -Denorm | -inf | NaN | |Dest - IEEE mode | +0 | +inf | -inf | -0 | NaN | |Dest - ALT mode | +fmax | -fmax | NaN | | Src | +inf | +0 | -0 | -inf | NaN | |Dest - IEEE mode | +0 | +inf | -inf | -0 | NaN |	Syntax: MATH_UNARY_REGIMM
0010b	LOG	Natural log: ln(src0)	Table:special value processing | Src | +inf | +0 / +Denorm | -0 / -Denorm | -inf | -F | NaN | |Dest - IEEE mode | +inf | -inf | -inf | NaN | NaN | NaN | NaN | |Dest - ALT mode | -fmax | -fmax | +F | NaN | | Src | +inf | +0	Syntax: MATH_UNARY_REGIMM

			| -0 | -inf | -F | NaN | |Dest - IEEE mode | +inf | -inf | -inf | NaN | NaN | NaN |	
0011b	EXP	Exponential (E^src0)	Table:special value processing | Src | +inf | +0 / +Denorm | -0 / -Denorm | -inf | -F | NaN | |Dest - IEEE mode | +inf | 1 | 1 | 0 | +F | NaN | |Dest - ALT mode | | 1 | 1 | | +F | NaN | | Src | +inf | +0 | -0 | -inf | -F | NaN | |Dest - IEEE mode | +inf | 1 | 1 | 0 | +F | NaN |	Syntax: MATH_UNARY_REGIMM
0100b	SQT	Square Root	Table:special value processing | Src | +inf | +0 / +Denorm | -0 / -Denorm | -inf | -F | NaN | |Dest - IEEE mode | +inf | 0 | -0 | NaN | NaN | NaN | |Dest - ALT mode | | 0 | 0 | | +F | NaN | | Src | +inf | +0 | -0 | -inf | -F | NaN | |Dest - IEEE mode | +inf | 0 | -0 | NaN | NaN | NaN |	Syntax: MATH_UNARY_REGIMM

			|	
0101b	RSQT	Reciprocal Square Root: 1/sqt(src)	Table:special value processing | Src | +inf | +0 / +Denorm | -0 / -Denorm | -inf | -F | NaN | |Dest - IEEE mode | +0 | +inf | -inf | NaN | NaN | NaN | |Dest - ALT mode | +fmax | +fmax | +F | NaN | | Src | +inf | +0 | -0 | -inf | -F | NaN | |Dest - IEEE mode | +0 | +inf | -inf | NaN | NaN | NaN |	Syntax: MATH_UNARY_REGIMM
0110b	SIN	Sine function. sin(src0)	Table:special value processing | Src | +inf | +0 / +Denorm | -0 / -Denorm | -inf | -F | NaN | |Dest - IEEE mode | NaN | +0 | -0 | NaN | -1 to 1 | NaN | |Dest - ALT mode | +0 | -0 | -1 to 1 | NaN | | Src | +inf | +0 | -0 | -inf | -F | NaN | |Dest - IEEE mode | NaN | +0 | -0 | NaN | -1 to 1 | NaN |	Syntax: MATH_UNARY_REGIMM
0111b	COS	Cosine function.	Table:special value	Syntax: MATH_UNARY_REGIMM

		cos(src0)	processing | Src | +inf | +0 / +Denorm | -0 / -Denorm | -inf | -F | NaN | |Dest - IEEE mode | NaN | +0 | -0 | NaN | -1 to 1 | NaN | |Dest - ALT mode | | +1 | +1 | -1 to 1 | NaN | | Src | +inf | +0 | -0 | -inf | -F | NaN | |Dest - IEEE mode | NaN | +0 | -0 | NaN | -1 to 1 | NaN |	
1000b	Reserved			
[1001b-1010b]	Reserved	Previously fdiv and pow. Fdiv x/y may be emulated via mul and inv: x*inv(y). Pow can be replaced via exp and log: X^Y = E^(Y*LOG(X))		
[1011b-1101b]	Reserved			
1110b	INVM	Reciprocal Macro for IEEE754-compliant fdiv		Syntax: MATH_MACRO_BINARY_REG_REG
1111b	RSQTM	Reciprocal Square Root Macro for IEEE754-compliant rsqt		Syntax: MATH_MACRO_UNARY_REG

MathMacroExt

MathMacroExt	
Source:	EuIsa
Size (in bits):	4
Specifies the extra registers used by the math macro instructions (formerly SpecialAcc).	

Value	Name
0000b	mme0 **[Default]**
0001b	mme1
0010b	mme2
0011b	mme3
0100b	mme4
0101b	mme5
0110b	mme6
0111b	mme7
1000b	nomme

Media Compression Format

Media Compression Format		
Size (in bits): 4		
Value	**Name**	**Description**
00001b	RGBA16_FLOAT	
00010b	Y210	
00011b	YUY2	
00100b	Y410 (10:10:10:2)	
00101b	Y216	
00110b	Y416	
00111b	P010	Luma P010 has MSB of 0 while chroma P010 has MSB of 1.
01000b	P016	Luma P016 has MSB of 0 while chroma P016 has MSB of 1.
01001b	AYUV	
01010b	ARGB 8b	
01011b	YCRCB_SwapY	
01100b	YCRCB_SwapUV	
01101b	YCRCB_SwapUVY	
01110b	RGB 10b	
01111b	NV21/NV12	Luma NV12 has MSB of 0 while chroma NV12 has MSB of 1.

Performance Counter Report Formats

Performance Counter Report Formats	
Size (in bits):	3

Value	Name
001b	
010b	
011b	
100b	
110b	
111b	

PredCtrl

PredCtrl	
Source:	EuIsa
Size (in bits):	4

Value	Name	Exists If
0000b	No Predication (normal) **[Default]**	
0001b	Sequential Flag Channel Mapping	
0010b	Replication swizzle .x	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
0010b	.anyv (any from f0.0-f1.0 on the same channel)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
0011b	Replication swizzle .y	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
0011b	.allv (all of f0.0-f1.0 on the same channel)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
0100b	Replication swizzle .z	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
0100b	.any2h (any in group of 2 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
0101b	Replication swizzle .w	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
0101b	.all2h (all in group of 2 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
0110b	.any4h	
0111b	.all4h	
1000b-1111b	Reserved	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
1000b	.any8h (any in group of 8 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1001b	.all8h (all in group of 8 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1010b	.any16h (any in group of 16 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1011b	.all16h (all in group of 16 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1100b	.any32h (any in group of 32 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1101b	.all32h (all in group of 32 channels)	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1110b-1111b	Reserved	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')

PREFERRED_SLM_SIZE

PREFERRED_SLM_SIZE - PREFERRED_SLM_SIZE

Size (in bits): 4

Specifies the preferred SLM size per subslice for this kernel.

Programming Notes

When bit 3 is not set in the enumerated value, the preferred SLM size per subslice is forced to **Max** to maintain backward compatibility with old drivers.

Restriction

Selected PREFERRED_SLM_SIZE must specify a size >= the selected SLM_SIZE

Value	Name	Description
0x0	Max **[Default]**	Preferred SLM size is the largest SLM size supported in the subslice.
[0x1-0x7]	Reserved	
0x8	0KB	Preferred SLM size is 0KB
0x9	16 KB	Preferred SLM size is 16KB
0xa	32 KB	Preferred SLM size is 32KB
0xb	64 KB	Preferred SLM size is 64KB
0xc	96 KB	Preferred SLM size is 96KB
0xd	128 KB	Preferred SLM size is 128KB

RegDataType

RegDataType	
Source:	EuIsa
Size (in bits):	4

Destination Type Numeric data type of the destination operand dst. The bits of the destination operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. For a send or sendc instruction, this field applies to CurrDst, the current destination operand. Three source instructions use a 3-bit encoding that allows fewer data types.

Value	Name	Description
0000b	:ub	Unsigned Byte (8-bit) Integer
0001b	:uw	Unsigned Word (16-bit) Integer
0010b	:ud	Unsigned DoubleWord (32-bit) Integer
0011b	:uq	Unsigned Quadword (64-bit) Integer
0100b	:b	Signed Byte (8-bit) Integer
0101b	:w	Signed Word (16-bit) Integer
0110b	:d	Signed DoubleWord (32-bit) Integer
0111b	:q	Signed QuadWord (64-bit) Integer
1000b	Reserved	
1001b	:hf	Half (16-bit) Float
1010b	:f	Single-Precision (32-bit) Float
1011b	:df	Double-Precision (64-bit) Float
1100b	Reserved	
1101b	:bf	Bfloat16. 16-bit floating point number with (1bit sign + 8bit exponent + 7bit mantissa)
1110b	Reserved	
1111b	Reserved	

Registers Per Thread

REGISTERS_PER_THREAD_SIZE - Registers Per Thread		
Size (in bits): 0		
Specifies the minimum number of registers allocated for each thread dispatch.		
Value	**Name**	**Description**
0	Default **[Default]**	For all threads except Compute thread, use 128 registers. For Compute Threads, use either 128 or 256 registers, based on STATE_COMPUTE_MODE.

RENDER_BARRIER_STAGE

RENDER_BARRIER_STAGE	
Size (in bits):	7
Stages a render barrier can either signal or wait.	

Value	Name
0x1	TOP
0x2	Color
0x4	Gpgpu
0x6	GPGPU and Color
0x10	Geom
0x20	Z
0x40	PS

RENDER_BARRIER_TYPE

RENDER_BARRIER_TYPE	
Size (in bits): 2	
Value	**Name**
0x1	Signal
0x2	Wait
0x3	Immediate

RenderCompressionFormat

CMP_FMT_RENDER_ACM - RenderCompressionFormat

Source: BSpec

Size (in bits): 5

This is Compression format encoding for Unified Lossless Compression

Value	Name	Description
00000b		**Main Surface Format Name** / **Casting** / **Compression Format Encoding** R32G32B32A32_FLOAT / 32b / 0x0 R32G32B32X32_FLOAT / 32b / 0x0 R32G32B32A32_SINT / 32b / 0x0
00001b		**Main Surface Format Name** / **Casting** / **Compression Format Encoding** R32G32B32A32_UINT / 32b / 0x1
00010b		**Main Surface Format Name** / **Casting** / **Compression Format Encoding** R32G32_FLOAT / 32b / 0x2 R32G32_SINT / 32b / 0x2
00011b		**Main Surface Format Name** / **Casting** / **Compression Format Encoding** R32G32_UINT / 32b / 0x3
00100b		**Main Surface Format Name** / **Casting** / **Compression Format Encoding** R16G16B16A16_UNORM / 16b / 0x4 R16G16B16X16_UNORM / 16b / 0x4 R16G16B16A16_UINT / 16b / 0x4
00101b		**Main Surface Format Name** / **Casting** / **Compression Format Encoding** R16G16B16A16_SNORM / 16b / 0x5 R16G16B16A16_SINT / 16b / 0x5 R16G16B16A16_FLOAT / 16b / 0x5 R16G16B16X16_FLOAT / 16b / 0x5
00110b		**Main Surface Format Name** / **Casting** / **Compression Format Encoding** R16G16_UNORM / 16b / 0x6 R16G16_UINT / 16b / 0x6
00111b		**Main Surface Format Name** / **Casting** / **Compression Format Encoding** R16G16_SNORM / 16b / 0x7 R16G16_SINT / 16b / 0x7 R16G16_FLOAT / 16b / 0x7

CMP_FMT_RENDER_ACM - RenderCompressionFormat

01000b

Main Surface Format Name	Casting	Compression Format Encoding
B8G8R8A8_UNORM	8b	0x8
B8G8R8X8_UNORM	8b	0x8
B8G8R8A8_UNORM_SRGB	8b	0x8
B8G8R8X8_UNORM_SRGB	8b	0x8
R8G8B8A8_UNORM	8b	0x8
R8G8B8X8_UNORM	8b	0x8
R8G8B8A8_UNORM_SRGB	8b	0x8
R8G8B8X8_UNORM_SRGB	8b	0x8
R8G8B8A8_UINT	8b	0x8

01001b

Main Surface Format Name	Casting	Compression Format Encoding
R8G8B8A8_SNORM	8b	0x9
R8G8B8A8_SINT	8b	0x9

01010b

Main Surface Format Name	Casting	Compression Format Encoding
B5G6R5_UNORM	8b	0xA
B5G6R5_UNORM_SRGB	8b	0xA
B5G5R5A1_UNORM	8b	0xA
B5G5R5A1_UNORM_SRGB	8b	0xA
B4G4R4A4_UNORM	8b	0xA
B4G4R4A4_UNORM_SRGB	8b	0xA
B5G5R5X1_UNORM	8b	0xA
B5G5R5X1_UNORM_SRGB	8b	0xA
A1B5G5R5_UNORM	8b	0xA
A4B4G4R4_UNORM	8b	0xA
R8G8_UNORM	8b	0xA
R8G8_UINT	8b	0xA

01011b

Main Surface Format Name	Casting	Compression Format Encoding
R8G8_SNORM	8b	0xB
R8G8_SINT	8b	0xB

01100b

Main Surface Format Name	Casting	Compression Format Encoding
R10G10B10A2_UNORM	8b	0xC
R10G10B10X2_UNORM	8b	0xC
R10G10B10A2_UNORM_SRGB	8b	0xC
R10G10B10_FLOAT_A2_UNORM	8b	0xC
R10G10B10A2_UINT	8b	0xC

CMP_FMT_RENDER_ACM - RenderCompressionFormat

		B10G10R10A2_UNORM	8b	0xC
		B10G10R10X2_UNORM	8b	0xC
		B10G10R10A2_UNORM_SRGB	8b	0xC
01101b		**Main Surface Format Name**	**Casting**	**Compression Format Encoding**
		R11G11B10_FLOAT	8b	0xD
01111b		**Main Surface Format Name**	**Casting**	**Compression Format Encoding**
		LINEAR DATA 16bpp component size INT/UINT/NORM/SNORM/FLOAT	16b	0xF
10000b		**Main Surface Format Name**	**Casting**	**Compression Format Encoding**
		R32_SINT	32b	0x10
		R32_FLOAT	32b	0x10
		D32_FLOAT	32b	0x10
10001b		**Main Surface Format Name**	**Casting**	**Compression Format Encoding**
		R32_UINT	32b	0x11
		D24_UNORM_X8	32b	0x11
10100b		**Main Surface Format Name**	**Casting**	**Compression Format Encoding**
		R16_UNORM	16b	0x14
		R16_UINT	16b	0x14
		D16_UNORM	16b	0x14
10101b		**Main Surface Format Name**	**Casting**	**Compression Format Encoding**
		R16_SNORM	16b	0x15
		R16_SINT	16b	0x15
		R16_FLOAT	16b	0x15
11000b		**Main Surface Format Name**	**Casting**	**Compression Format Encoding**
		R8_UNORM	8b	0x18
		R8_UINT	8b	0x18
		A8_UNORM	8b	0x18
11001b		**Main Surface Format Name**	**Casting**	**Compression Format Encoding**
		R8_SNORM	8b	0x19
		R8_SINT	8b	0x19
11111b		**Main Surface Format Name**	**Casting**	**Compression Format Encoding**
		LINEAR DATA Raw, 8 or 32bpp component size INT/UINT/NORM/SNORM/FLOAT	8b	0x1F

Render Compression Format

CMP_FMT_RENDER_ATS - Render Compression Format

Size (in bits): 3

<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>00000b</td><td></td><td>
<table>
<tr><th>Format</th><th>Casting or Sample Ratio</th><th>Encoding</th></tr>
<tr><td>RGBA16_FLOAT</td><td>16b</td><td>0</td></tr>
<tr><td>R16G16B16A16_UNORM</td><td>16b</td><td>0</td></tr>
<tr><td>R16G16B16A16_SNORM</td><td>16b</td><td>0</td></tr>
<tr><td>R16G16B16A16_SINT</td><td>16b</td><td>0</td></tr>
<tr><td>R16G16B16A16_UINT</td><td>16b</td><td>0</td></tr>
<tr><td>R16G16B16A16_FLOAT</td><td>16b</td><td>0</td></tr>
<tr><td>R16G16B16X16_FLOAT</td><td>16b</td><td>0</td></tr>
<tr><td>R16G16_UNORM</td><td>16b</td><td>0</td></tr>
<tr><td>R16G16_SNORM</td><td>16b</td><td>0</td></tr>
<tr><td>R16G16_SINT</td><td>16b</td><td>0</td></tr>
<tr><td>R16G16_UINT</td><td>16b</td><td>0</td></tr>
<tr><td>R16G16_FLOAT</td><td>16b</td><td>0</td></tr>
<tr><td>R16_UNORM</td><td>16b</td><td>0</td></tr>
<tr><td>R16_UINT</td><td>16b</td><td>0</td></tr>
<tr><td>D16_UNORM</td><td>16b</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>00001b</td><td></td><td>
<table>
<tr><th>Format</th><th>Casting or Sample Ratio</th><th>Encoding</th></tr>
<tr><td>R32G32B32A32_FLOAT</td><td>32b</td><td>1</td></tr>
<tr><td>R32G32B32X32_FLOAT</td><td>32b</td><td>1</td></tr>
<tr><td>R32G32B32A32_SINT</td><td>32b</td><td>1</td></tr>
<tr><td>R32G32B32A32_UINT</td><td>32b</td><td>1</td></tr>
<tr><td>R32G32_FLOAT</td><td>32b</td><td>1</td></tr>
<tr><td>R32G32_SINT</td><td>32b</td><td>1</td></tr>
<tr><td>R32G32_UINT</td><td>32b</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>00010b</td><td></td><td>
<table>
<tr><th>Format</th><th>Casting or Sample Ratio</th><th>Encoding</th></tr>
<tr><td>B8G8R8A8_UNORM</td><td>8b</td><td>2</td></tr>
<tr><td>B8G8R8A8_UNORM_SRGB</td><td>8b</td><td>2</td></tr>
<tr><td>R8G8B8A8_UNORM</td><td>8b</td><td>2</td></tr>
<tr><td>R8G8B8A8_UNORM_SRGB</td><td>8b</td><td>2</td></tr>
<tr><td>R8G8B8A8_SNORM</td><td>8b</td><td>2</td></tr>
<tr><td>R8G8B8A8_SINT</td><td>8b</td><td>2</td></tr>
</table>
</td></tr>
</table>

		R8G8B8A8_UINT	8b	2
		R32_UINT	32b	2
		B5G6R5_UNORM	8b	2
		B5G6R5_UNORM_SRGB	8b	2
		B5G5R5A1_UNORM	8b	2
		B5G5R5A1_UNORM_SRGB	8b	2
		B4G4R4A4_UNORM	8b	2
		B4G4R4A4_UNORM_SRGB	8b	2
		R8G8_UNORM	8b	2
		R8G8_SNORM	8b	2
		R8G8_SINT	8b	2
		R8G8_UINT	8b	2
		B5G5R5X1_UNORM	8b	2
		B5G5R5X1_UNORM_SRGB	8b	2
		A1B5G5R5_UNORM	8b	2
		A4B4G4R4_UNORM	8b	2
		R8_UNORM	8b	2
		R8_UINT	8b	2
		A8_UNORM	8b	2
		D24UNORMX8	32b	2
		D32FLOAT	32b	2

00011b

Format	Casting or Sample Ratio	Encoding
R10G10B10A2_UNORM	8b	3
R10G10B10A2_UNORM_SRGB	8b	3
R10G10B10_FLOAT_A2_UNORM	8b	3
R10G10B10A2_UINT	8b	3
B10G10R10A2_UNORM	8b	3
B10G10R10A2_UNORM_SRGB	8b	3

00100b

Format	Casting or Sample Ratio	Encoding
R11G11B10_FLOAT	8b	4

00101b

Format	Casting or Sample Ratio	Encoding
R32_SINT	32b	5
R32_FLOAT	32b	5

00110b

Format	Casting or Sample Ratio	Encoding
R16_SNORM	16b	6

CMP_FMT_RENDER_ATS - Render Compression Format

		R16_SINT, 16b, 6 R16_FLOAT, 16b, 6
00111b		**Format** / **Casting or Sample Ratio** / **Encoding** R8_SNORM, 8b, 7 R8_SINT, 8b, 7

RepeatCount

RepeatCount	
Source:	Eulsa
Size (in bits):	3
This field indicate the repeat count of the instruction.	

Value	Name
000b	1
001b	2
010b	3
011b	4
100b	5
101b	6
110b	7
111b	8

Sampler State Count

SAMPLER_STATE_COUNT - Sampler State Count
Size (in bits): 3
Specifies how many samplers (in multiples of 4) the kernel uses. Used only for prefetching the associated sampler state entries.

Programming Notes
Typically set to 0 to avoid prefetching on every thread dispatch.

Value	Name
0h	No samplers used **[Default]**
1h	Between 1 and 4 samplers used
2h	Between 5 and 8 samplers used
3h	Between 9 and 12 samplers used
4h	Between 13 and 16 samplers used

Saturate

Saturate	
Size (in bits):	1
This field controls the destination saturation. When it is set, output data to the destination register are saturated. The saturation operation depends on the destination data type. Saturation is the operation that converts any data that is outside the saturation target range for the data type to the closest representable value with the target range. If destination type is float, saturation target range is [0, 1]. For example, any positive number greater than 1 (including +INF) is saturated to 1 and any negative number (including -INF) is saturated to 0. A NaN is saturated to 0, For integer data types, the maximum range for the given numerical data type is the saturation target range. When it is not set, output data to the destination register are not saturated. For example, a wrapped result (modular) is output to the destination for an overflowed integer data. More details can be found in the Data Types chapter.	

Value	Name
0	No Destination modification **[Default]**
1	Saturate Destination

SFID

SFID	
Source:	EuIsa
Size (in bits):	4

The following table lists the assignments (encodings) of the Shared Function and Fixed Function IDs used within the GPE. A Shared Function is a valid target of a message initiated via a 'send' instruction. A Fixed Function is an identifiable unit of the 3D or Media pipeline. Note that the Thread Spawner is both a Shared Function and Fixed Function. Note: The initial intention was to combine these two ID namespaces, so that (theoretically) an agent (such as the Thread Spawner) that served both as a Shared Function and Fixed Function would have a single, unique 4-bit ID encoding. However, this combination is not a requirement of the architecture.

Programming Notes
SFID_DP_DC1 is an extension of SFID_DP_DC0 to allow for more message types. They act as a single logical entity.
SFID_DP_DC1 and SFID_DP_DC2 are extensions of SFID_DP_DC0 to allow for more message types. They act as a single logical entity.

Value	Name	Description	Properties
0000b	SFID_NULL	Null	Syntax: SEND_BINARY
0001b	SFID_UGML	Low-Bandwidth Untyped Global Memory	Syntax: SEND_BINARY
0010b	SFID_SAMPLER	Sampler	Syntax: SEND_BINARY
0011b	SFID_GATEWAY	Message Gateway	Syntax: SEND_BINARY
0100b	SFID_DP_DC2	Data Cache Data Port 2	Syntax: SEND_BINARY
0101b	SFID_DP_RC	Render Cache Data Port	Syntax: SEND_BINARY
0110b	SFID_URB	URB	Syntax: SEND_BINARY
0111b	SFID_BTD	Bindless Thread Dispatcher	Syntax: SEND_BINARY
1000b	SFID_RTA	Ray Trace Accelerator	Syntax: SEND_BINARY
1001b	SFID_DP_DCRO	Data Cache Read Only Data Port	Syntax: SEND_BINARY
1010b	SFID_DP_DC0	Data Cache Data Port	Syntax: SEND_BINARY
1011b	SFID_PI	Pixel Interpolator	Syntax: SEND_BINARY
1100b	SFID_DP_DC1	Data Cache Data Port 1	Syntax: SEND_BINARY
1101b	SFID_TGM	Typed Global Memory	Syntax: SEND_BINARY
1110b	SFID_SLM	Shared Local Memory	Syntax: SEND_BINARY
1111b	SFID_UGM	Untyped Global Memory	Syntax: SEND_BINARY

Shader Channel Select

Shader Channel Select		
Size (in bits): 3		
Value	**Name**	**Description**
0	ZERO	
1	ONE	
2	Reserved	
3	Reserved	
4	RED	Shader channel is set to surface red channel
5	GREEN	Shader channel is set to surface green channel
6	BLUE	Shader channel is set to surface blue channel
7	ALPHA	Shader channel is set to surface alpha channel

Shared Local Memory Size

SLM_SIZE - Shared Local Memory Size		
Size (in bits):	3	
This field indicates how much Shared Local Memory the thread group requires.		
Value	**Name**	**Description**
0	Encodes 0KB	No SLM used
0x1	Encodes 1KB	
0x2	Encodes 2KB	
0x3	Encodes 4KB	
0x4	Encodes 8KB	
0x5	Encodes 16KB	
0x6	Encodes 32KB	
0x7	Encodes 64KB	

SIMD Mode

SIMD Mode	
Size (in bits):	3

Value	Name
0	SIMD8 + Integer Return
1	SIMD8
2	SIMD16
3	Reserved2
4	SIMD16 + Integer Return
5	SIMD8H
6	SIMD16H
7	Reserved7

Slice Hash Control

Slice Hash Control	
Source:	RenderCS
Size (in bits):	2

This is a sample showing the basic structure of an explicit enumeration.

Value	Name	Description
00b	Computed	Use Computed pixelhash_id
01b	Unbalanced table[0]	Use Computed pixelhash_id when balanced, Table[0] when unbalanced
10b	Table[0]	Use Table[0]
11b	Table[1]	Use Table[1]

SrcMod

SrcMod	
Source:	EuIsa
Size (in bits):	2

Source Modifier This field specifies the numeric modification of a source operand. The value of each data element of a source operand can optionally have its absolute value taken and/or its sign inverted prior to delivery to the execution pipe. The absolute value is prior to negate such that a guaranteed negative value can be produced. This field only applies to source operand. It does not apply to destination. This field is not present for an immediate source operand.

When used with logic instructions (and, not, or, xor), this field indicates whether the source bits are inverted (bitwise NOT) before delivery to the execution pipe, regardless of the source type.

Value	Name	Description
00b	No modification	
01b	abs	Absolute value Logic instructions: No modification (This encoding cannot be selected in the assembler syntax)
10b	negate	Negate Logic instructions: Bitwise NOT, inverting the source bits
11b	negate of abs	Negate of the absolute (forced negative value) Logic instructions: No modification (This encoding cannot be selected in the assembler syntax)

SubBytePrecision

SubBytePrecision			
Size (in bits): 2			
This field provide the sub-byte level precision when operand is smaller that what can be specified in the TernaryDataType field. E.g. s8 would indicate that a dword is chunked into 4 x 8b signed values; u2 would indicate that the operand, a DWORD is broken into 16 x 2b unsigned values.			
Value	**Name**	**Description**	**Exists If**
00b	None	This value indicates that sub-byte information is not used.	
01b	:u4	This value encodes a 4bit unsigned integer value.	([TernaryDataType]==':ub')
01b	:s4	This value encodes a 4bit signed integer value.	([TernaryDataType]==':b')
10b	:u2	This value encodes a 2bit unsigned integer value.	([TernaryDataType]==':ub')
10b	:s2	This value encodes a 2bit signed integer value.	([TernaryDataType]==':b')
11b	Reserved		([TernaryDataType]==':ub') OR ([TernaryDataType]==':b')
[01b-11b]	Reserved		([TernaryDataType]!=':ub') AND ([TernaryDataType]!=':b')

SURFACE_FORMAT

SURFACE_FORMAT

Size (in bits): 9

The following table indicates the supported surface formats and the 9-bit encoding for each. Note that some of these formats are used not only by the Sampling Engine, but also by the Data Port and the Vertex Fetch unit.

Value	Name	Description
000h	R32G32B32A32_FLOAT	
001h	R32G32B32A32_SINT	
002h	R32G32B32A32_UINT	
003h	R32G32B32A32_UNORM	
004h	R32G32B32A32_SNORM	
005h	R64G64_FLOAT	
006h	R32G32B32X32_FLOAT	
007h	R32G32B32A32_SSCALED	
008h	R32G32B32A32_USCALED	
020h	R32G32B32A32_SFIXED	
021h	R64G64_PASSTHRU	
040h	R32G32B32_FLOAT	
041h	R32G32B32_SINT	
042h	R32G32B32_UINT	
043h	R32G32B32_UNORM	
044h	R32G32B32_SNORM	
045h	R32G32B32_SSCALED	
046h	R32G32B32_USCALED	
050h	R32G32B32_SFIXED	
080h	R16G16B16A16_UNORM	
081h	R16G16B16A16_SNORM	
082h	R16G16B16A16_SINT	
083h	R16G16B16A16_UINT	
084h	R16G16B16A16_FLOAT	
085h	R32G32_FLOAT	
086h	R32G32_SINT	
087h	R32G32_UINT	
088h	R32_FLOAT_X8X24_TYPELESS	
089h	X32_TYPELESS_G8X24_UINT	
08Ah	L32A32_FLOAT	

SURFACE_FORMAT

08Bh	R32G32_UNORM	
08Ch	R32G32_SNORM	
08Dh	R64_FLOAT	
08Eh	R16G16B16X16_UNORM	
08Fh	R16G16B16X16_FLOAT	
090h	A32X32_FLOAT	
091h	L32X32_FLOAT	
092h	I32X32_FLOAT	
093h	R16G16B16A16_SSCALED	
094h	R16G16B16A16_USCALED	
095h	R32G32_SSCALED	
096h	R32G32_USCALED	
0A0h	R32G32_SFIXED	
0A1h	R64_PASSTHRU	
0C0h	B8G8R8A8_UNORM	
0C1h	B8G8R8A8_UNORM_SRGB	
0C2h	R10G10B10A2_UNORM	
0C3h	R10G10B10A2_UNORM_SRGB	
0C4h	R10G10B10A2_UINT	
0C5h	R10G10B10_SNORM_A2_UNORM	
0C7h	R8G8B8A8_UNORM	
0C8h	R8G8B8A8_UNORM_SRGB	
0C9h	R8G8B8A8_SNORM	
0CAh	R8G8B8A8_SINT	
0CBh	R8G8B8A8_UINT	
0CCh	R16G16_UNORM	
0CDh	R16G16_SNORM	
0CEh	R16G16_SINT	
0CFh	R16G16_UINT	
0D0h	R16G16_FLOAT	
0D1h	B10G10R10A2_UNORM	
0D2h	B10G10R10A2_UNORM_SRGB	
0D3h	R11G11B10_FLOAT	
0D5h	R10G10B10_FLOAT_A2_UNORM	
0D6h	R32_SINT	
0D7h	R32_UINT	

SURFACE_FORMAT		
0D8h	R32_FLOAT	
0D9h	R24_UNORM_X8_TYPELESS	
0DAh	X24_TYPELESS_G8_UINT	
0DDh	L32_UNORM	
0DEh	A32_UNORM	
0DFh	L16A16_UNORM	
0E0h	I24X8_UNORM	
0E1h	L24X8_UNORM	
0E2h	A24X8_UNORM	
0E3h	I32_FLOAT	
0E4h	L32_FLOAT	
0E5h	A32_FLOAT	
0E6h	X8B8_UNORM_G8R8_SNORM	
0E7h	A8X8_UNORM_G8R8_SNORM	
0E8h	B8X8_UNORM_G8R8_SNORM	
0E9h	B8G8R8X8_UNORM	
0EAh	B8G8R8X8_UNORM_SRGB	
0EBh	R8G8B8X8_UNORM	
0ECh	R8G8B8X8_UNORM_SRGB	
0EDh	R9G9B9E5_SHAREDEXP	
0EEh	B10G10R10X2_UNORM	
0F0h	L16A16_FLOAT	
0F1h	R32_UNORM	
0F2h	R32_SNORM	
0F3h	R10G10B10X2_USCALED	
0F4h	R8G8B8A8_SSCALED	
0F5h	R8G8B8A8_USCALED	
0F6h	R16G16_SSCALED	
0F7h	R16G16_USCALED	
0F8h	R32_SSCALED	
0F9h	R32_USCALED	
100h	B5G6R5_UNORM	
101h	B5G6R5_UNORM_SRGB	
102h	B5G5R5A1_UNORM	
103h	B5G5R5A1_UNORM_SRGB	
104h	B4G4R4A4_UNORM	

SURFACE_FORMAT		
105h	B4G4R4A4_UNORM_SRGB	
106h	R8G8_UNORM	
107h	R8G8_SNORM	
108h	R8G8_SINT	
109h	R8G8_UINT	
10Ah	R16_UNORM	
10Bh	R16_SNORM	
10Ch	R16_SINT	
10Dh	R16_UINT	
10Eh	R16_FLOAT	
10Fh	A8P8_UNORM_PALETTE0	
110h	A8P8_UNORM_PALETTE1	
111h	I16_UNORM	
112h	L16_UNORM	
113h	A16_UNORM	
114h	L8A8_UNORM	
115h	I16_FLOAT	
116h	L16_FLOAT	
117h	A16_FLOAT	
118h	L8A8_UNORM_SRGB	
119h	R5G5_SNORM_B6_UNORM	
11Ah	B5G5R5X1_UNORM	
11Bh	B5G5R5X1_UNORM_SRGB	
11Ch	R8G8_SSCALED	
11Dh	R8G8_USCALED	
11Eh	R16_SSCALED	
11Fh	R16_USCALED	
122h	P8A8_UNORM_PALETTE0	
123h	P8A8_UNORM_PALETTE1	
124h	A1B5G5R5_UNORM	
125h	A4B4G4R4_UNORM	
126h	L8A8_UINT	
127h	L8A8_SINT	
140h	R8_UNORM	
141h	R8_SNORM	
142h	R8_SINT	

SURFACE_FORMAT

143h	R8_UINT	
144h	A8_UNORM	
145h	I8_UNORM	
146h	L8_UNORM	
147h	P4A4_UNORM_PALETTE0	
148h	A4P4_UNORM_PALETTE0	
149h	R8_SSCALED	
14Ah	R8_USCALED	
14Bh	P8_UNORM_PALETTE0	
14Ch	L8_UNORM_SRGB	
14Dh	P8_UNORM_PALETTE1	
14Eh	P4A4_UNORM_PALETTE1	
14Fh	A4P4_UNORM_PALETTE1	
150h	Y8_UNORM	
152h	L8_UINT	
153h	L8_SINT	
154h	I8_UINT	
155h	I8_SINT	
180h	DXT1_RGB_SRGB	
181h	R1_UNORM	SET0_LEGACY: Undefined behavior if used in any feature added. See Legacy sampler feature page for details
182h	YCRCB_NORMAL	
183h	YCRCB_SWAPUVY	
184h	P2_UNORM_PALETTE0	
185h	P2_UNORM_PALETTE1	
186h	BC1_UNORM	(DXT1)
187h	BC2_UNORM	(DXT2/3)
188h	BC3_UNORM	(DXT4/5)
189h	BC4_UNORM	
18Ah	BC5_UNORM	
18Bh	BC1_UNORM_SRGB	(DXT1_SRGB)
18Ch	BC2_UNORM_SRGB	(DXT2/3_SRGB)
18Dh	BC3_UNORM_SRGB	(DXT4/5_SRGB)
18Eh	MONO8	SET0_LEGACY: Undefined behavior if used in any feature added. See Legacy sampler feature page for details
18Fh	YCRCB_SWAPUV	
190h	YCRCB_SWAPY	

SURFACE_FORMAT

191h	DXT1_RGB	
192h	RESERVED_192	This value is reserved for internal use.
193h	R8G8B8_UNORM	
194h	R8G8B8_SNORM	
195h	R8G8B8_SSCALED	
196h	R8G8B8_USCALED	
197h	R64G64B64A64_FLOAT	
198h	R64G64B64_FLOAT	
199h	BC4_SNORM	
19Ah	BC5_SNORM	
19Bh	R16G16B16_FLOAT	
19Ch	R16G16B16_UNORM	
19Dh	R16G16B16_SNORM	
19Eh	R16G16B16_SSCALED	
19Fh	R16G16B16_USCALED	
1A1h	BC6H_SF16	
1A2h	BC7_UNORM	
1A3h	BC7_UNORM_SRGB	
1A4h	BC6H_UF16	
1A5h	PLANAR_420_8	
1A6h	PLANAR_420_16	
1A8h	R8G8B8_UNORM_SRGB	
1A9h	ETC1_RGB8	
1AAh	ETC2_RGB8	
1ABh	EAC_R11	
1ACh	EAC_RG11	
1ADh	EAC_SIGNED_R11	
1AEh	EAC_SIGNED_RG11	
1AFh	ETC2_SRGB8	
1B0h	R16G16B16_UINT	
1B1h	R16G16B16_SINT	
1B2h	R32_SFIXED	
1B3h	R10G10B10A2_SNORM	
1B4h	R10G10B10A2_USCALED	
1B5h	R10G10B10A2_SSCALED	
1B6h	R10G10B10A2_SINT	

SURFACE_FORMAT		
1B7h	B10G10R10A2_SNORM	
1B8h	B10G10R10A2_USCALED	
1B9h	B10G10R10A2_SSCALED	
1BAh	B10G10R10A2_UINT	
1BBh	B10G10R10A2_SINT	
1BCh	R64G64B64A64_PASSTHRU	
1BDh	R64G64B64_PASSTHRU	
1C0h	ETC2_RGB8_PTA	
1C1h	ETC2_SRGB8_PTA	
1C2h	ETC2_EAC_RGBA8	
1C3h	ETC2_EAC_SRGB8_A8	
1C8h	R8G8B8_UINT	
1C9h	R8G8B8_SINT	
1FFh	RAW	

SyncFC

SyncFC

Source: EuIsa

Size (in bits): 4

Subfunctions that the sync instruction supports.

Value	Name	Description	Properties
0000b	No Operation	Performs no operation. Regular SWSB constraints are checked.	Syntax: SYNC_UNARY
0010b	SBID Read Wait	Blocks until pending out-of-order source accesses are complete. If a mask is provided as immediate value in src0 then specific SBID resources can be checked, else all SBID resources are checked for source access complete status.	Syntax: SYNC_UNARY
0011b	SBID Write Wait	Blocks until pending out-of-order writebacks are complete.If a mask is provided as immediate value in src0 then specific SBID resources can be checked, else all SBID resources are checked for writeback complete status.	Syntax: SYNC_UNARY
1110b	Wait on Barrier	[] Blocks until the notification count reaches 0. The wait instruction evaluates the value of the notification count register nreg. If nreg is zero, thread execution is suspended and the thread is put in 'wait_for_notification' state. If nreg is not zero (i.e., one or more notifications have been received), nreg is decremented by one and the thread continues executing on the next instruction. If a thread is in the 'wait_for_notification' state, when a notification arrives, the notification count register is incremented by one. As the notification count register becomes nonzero, the thread wakes up to continue execution and at the same time the notification register is decremented by one. If only one notification arrived, the notification register value becomes zero. However, during the above mentioned time period, it is possible that more notifications may arrive, making the notification register nonzero again. This operation implicitly accesses n0 (typically n0.0 for barriers).	Syntax: SYNC_UNARY
1111b	Wait on Host Notification	Similar to .bar, but waits the host's notification register (typically n0.1). See that element for more information.	Syntax: SYNC_UNARY

SystolicFC

SystolicFC	
Source:	EuIsa
Size (in bits):	5

The various systolic depths and repeat counts dpas can encode. Some projects may only enable a subset of these.

Value	Name	Description	Properties
00000b	Depth 1x1	Uses a systolic depth of 1 with a repeat count of 1.	Syntax: TERNARY_REGIMM_REG_REGIMM
00001b	Depth 1x2	Uses a systolic depth of 1 with a repeat count of 2.	Syntax: TERNARY_REGIMM_REG_REGIMM
00010b	Depth 1x3	Uses a systolic depth of 1 with a repeat count of 3.	Syntax: TERNARY_REGIMM_REG_REGIMM
00011b	Depth 1x4	Uses a systolic depth of 1 with a repeat count of 4.	Syntax: TERNARY_REGIMM_REG_REGIMM
00100b	Depth 1x5	Uses a systolic depth of 1 with a repeat count of 5.	Syntax: TERNARY_REGIMM_REG_REGIMM
00101b	Depth 1x6	Uses a systolic depth of 1 with a repeat count of 6.	Syntax: TERNARY_REGIMM_REG_REGIMM
00110b	Depth 1x7	Uses a systolic depth of 1 with a repeat count of 7.	Syntax: TERNARY_REGIMM_REG_REGIMM
00111b	Depth 1x8	Uses a systolic depth of 1 with a repeat count of 8.	Syntax: TERNARY_REGIMM_REG_REGIMM
01000b	Depth 2x1	Uses a systolic depth of 2 with a repeat count of 1.	Syntax: TERNARY_REGIMM_REG_REGIMM
01001b	Depth 2x2	Uses a systolic depth of 2 with a repeat count of 2.	Syntax: TERNARY_REGIMM_REG_REGIMM
01010b	Depth 2x3	Uses a systolic depth of 2 with a repeat count of 3.	Syntax: TERNARY_REGIMM_REG_REGIMM
01011b	Depth 2x4	Uses a systolic depth of 2 with a repeat count of 4.	Syntax: TERNARY_REGIMM_REG_REGIMM
01100b	Depth 2x5	Uses a systolic depth of 2 with a repeat count of 5.	Syntax: TERNARY_REGIMM_REG_REGIMM
01101b	Depth 2x6	Uses a systolic depth of 2 with a repeat count of 6.	Syntax: TERNARY_REGIMM_REG_REGIMM
01110b	Depth 2x7	Uses a systolic depth of 2 with a repeat count of 7.	Syntax: TERNARY_REGIMM_REG_REGIMM
01111b	Depth 2x8	Uses a systolic depth of 2 with a repeat count of 8.	Syntax: TERNARY_REGIMM_REG_REGIMM

SystolicFC			
10000b	Depth 4x1	Uses a systolic depth of 4 with a repeat count of 1.	Syntax: TERNARY_REGIMM_REG_REGIMM
10001b	Depth 4x2	Uses a systolic depth of 4 with a repeat count of 2.	Syntax: TERNARY_REGIMM_REG_REGIMM
10010b	Depth 4x3	Uses a systolic depth of 4 with a repeat count of 3.	Syntax: TERNARY_REGIMM_REG_REGIMM
10011b	Depth 4x4	Uses a systolic depth of 4 with a repeat count of 4.	Syntax: TERNARY_REGIMM_REG_REGIMM
10100b	Depth 4x5	Uses a systolic depth of 4 with a repeat count of 5.	Syntax: TERNARY_REGIMM_REG_REGIMM
10101b	Depth 4x6	Uses a systolic depth of 4 with a repeat count of 6.	Syntax: TERNARY_REGIMM_REG_REGIMM
10110b	Depth 4x7	Uses a systolic depth of 4 with a repeat count of 7.	Syntax: TERNARY_REGIMM_REG_REGIMM
10111b	Depth 4x8	Uses a systolic depth of 4 with a repeat count of 8.	Syntax: TERNARY_REGIMM_REG_REGIMM
11000b	Depth 8x1	Uses a systolic depth of 8 with a repeat count of 1.	Syntax: TERNARY_REGIMM_REG_REGIMM
11001b	Depth 8x2	Uses a systolic depth of 8 with a repeat count of 2.	Syntax: TERNARY_REGIMM_REG_REGIMM
11010b	Depth 8x3	Uses a systolic depth of 8 with a repeat count of 3.	Syntax: TERNARY_REGIMM_REG_REGIMM
11011b	Depth 8x4	Uses a systolic depth of 8 with a repeat count of 4.	Syntax: TERNARY_REGIMM_REG_REGIMM
11100b	Depth 8x5	Uses a systolic depth of 8 with a repeat count of 5.	Syntax: TERNARY_REGIMM_REG_REGIMM
11101b	Depth 8x6	Uses a systolic depth of 8 with a repeat count of 6.	Syntax: TERNARY_REGIMM_REG_REGIMM
11110b	Depth 8x7	Uses a systolic depth of 8 with a repeat count of 7.	Syntax: TERNARY_REGIMM_REG_REGIMM
11111b	Depth 8x8	Uses a systolic depth of 8 with a repeat count of 8.	Syntax: TERNARY_REGIMM_REG_REGIMM

TernaryDataType

TernaryDataType			
Source:	EuIsa		
Size (in bits):	3		
This field provide the datatype of the source and destination operands for ternary instruction.			
Value	**Name**	**Description**	**Exists If**
000b	:ub	Unsigned Byte (8-bit) Integer	(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Integer)
001b	:uw	Unsigned Word (16-bit) Integer	(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Integer)
010b	:ud	Unsigned DoubleWord (32-bit) Integer	(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Integer)
011b	:uq	Unsigned Quadword (64-bit) Integer	(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Integer)
100b	:b	Signed Byte (8-bit) Integer	(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Integer)
101b	:w	Signed Word (16-bit) Integer	(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Integer)
110b	:d	Signed DoubleWord (32-bit) Integer	(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Integer)
111b	:q	Signed QuadWord (64-bit) Integer	(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Integer)
000b	Reserved		(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Float)
001b	:hf	Half (16-bit) Float	(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Float)
010b	:f	Single-Precision (32-bit) Float	(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Float)
011b	:df	Double-Precision (64-bit) Float	(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Float)
100b	Reserved		(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Float)
101b	:bf	Bfloat16. 16-bit floating point number with (1bit sign + 8bit exponent + 7bit mantissa).	(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Float)
110b	Reserved		(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Float)
111b	Reserved		(Structure[EU_INSTRUCTION_BASIC_THREE_SRC][ExecDataType]==Float)

TernaryVertStride

TernaryVertStride	
Source:	EuIsa
Size (in bits):	2
Source Vertical Stride is required for regioning/accessing datatypes of varied size. It is one way to obtain a vector of scalars.	

Value	Name
00b	0 - elements
01b	1 - elements
10b	4 - elements
11b	8 - elements

Texture Coordinate Mode

Texture Coordinate Mode		
Size (in bits): 3		
Value	**Name**	**Description**
0h	WRAP	Map is repeated in the U direction
1h	MIRROR	Map is mirrored in the U direction
2h	CLAMP	Map is clamped to the edges of the accessed map
3h	CUBE	For cube-mapping, filtering in edges access adjacent map faces
4h	CLAMP_BORDER	Map is infinitely extended with the border color
5h	MIRROR_ONCE	Map is mirrored once about origin, then clamped
6h	HALF_BORDER	Map is infinitely extended with the average of the nearest edge texel and the border color
7h	MIRROR_101	Map is mirrored one time in each direction, but the first pixel of the reflected image is skipped, and the reflected image is effectively 1 pixel less in that direction. May only be used on 2D surfaces.

VertStride

VertStride	
Source:	EuIsa
Size (in bits):	4

Description
Vertical Stride. The field provides the vertical stride of the register region in unit of data elements for an operand. Encoding of this field provides values of 0 or powers of 2, ranging from 1 to 32 elements. Larger values are not supported due to the restriction that a source operand must reside within two adjacent 256-bit registers (64 bytes total). Special encoding 1111b (0xF) is only valid when the operand is in register-indirect addressing mode (AddrMode = 1). If this field is set to 0xF, one or more sub-registers of the address registers may be used to compute the addresses. Each address sub-register provides the origin for a row of data element. The number of address sub-registers used is determined by the division of ExecSize of the instruction by the Width fields of the operand. This field only applies to source operand. It does not apply to destination. This field is not present for an immediate source operand.

Programming Notes
Note 1: If indirect address is supported for src1, encoding 0xF is reserved for src1 - only single-index indirect addressing is supported.
Note 2: Encoding 0010 applies for QWord-size operands.

Value	Name	Programming Notes
0000b	0 elements	
0001b	1 element	Align1 mode only.
0010b	2 elements	
0011b	4 elements	
0100b	8 elements	Align1 mode only.
0101b	16 elements	Applies to byte or word operand only. Align1 mode only.
0110b	32 elements	Applies to byte operand only. Align1 mode only.
0111b-1110b	Reserved	
1111b	VxH or Vx1 mode	Only valid for register-indirect addressing in Align1 mode.

Width

Width	
Source:	EuIsa
Size (in bits):	3

This field specifies the number of elements in the horizontal dimension of the region for a source operand. This field cannot exceed the ExecSize field of the instruction. This field only applies to source operand. It does not apply to destination. This field is not present for an immediate source operand.

Programming Notes
Note that with ExecSize of 32, because the maximum Width is 16, there are at least two rows in a source region.

Value	Name
000b	1 elements
001b	2 elements
010b	4 elements
011b	8 elements
100b	16 elements
101b-111b	Reserved

WRAP_SHORTEST_ENABLE

WRAP_SHORTEST_ENABLE

Source: RenderCS

Size (in bits): 4

This state selects which components (if any) of Attribute [n] are to be interpolated in a "wrap shortest" fashion. Operation is UNDEFINED if any of these bits are set and the Constant Interpolation Enable bit associated with this attribute is set. Note that wrap-shortest interpolation is only supported for Attributes 0-15.

Value	Name	Description
0001b	X	Wrap Shortest X Component
0010b	Y	Wrap Shortest Y Component
0011b	XY	Wrap Shortest XY Components
0100b	Z	Wrap Shortest Z Component
0101b	XZ	Wrap Shortest XZ Components
0110b	YZ	Wrap Shortest YZ Components
0111b	XYZ	Wrap Shortest XYZ Components
1000b	W	Wrap Shortest W Component
1001b	XW	Wrap Shortest XW Components
1010b	YW	Wrap Shortest YW Components
1011b	XYW	Wrap Shortest XYW Components
1100b	ZW	Wrap Shortest ZW Components
1101b	XZW	Wrap Shortest XZW Components
1110b	YZW	Wrap Shortest YZW Components
1111b	XYZW	Wrap Shortest XYZW Components