Intel® Advanced Performance Extensions (Intel® APX)

Assembly Syntax Recommendations

March 1, 2024
Revision 1.0

Document Number: 817241-001US

Notices & **Disclaimers**

This document contains information on products in the design phase of development. The information here is subject to change without notice. Do not finalize a design with this information.

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

No product or component can be absolutely secure.

Your costs and results may vary.

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.

All product plans and roadmaps are subject to change without notice. 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.

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.

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.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that a) you may publish an unmodified copy and b) code included in this document is licensed subject to the Zero-Clause BSD open source license (0BSD), https://opensource.org/licenses/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.

Contents

Intel® Advanced Performance Extensions (Intel® APX) introduces several new concepts that require new notations in assembly syntax. This document discusses recommendations for the new notations.

There are two main families of assembly syntax for x86: Intel syntax and AT&T syntax. This document provides recommendations for both. In all examples below, each pair of Intel-syntax and AT&T-syntax assembly instructions given on the same line denote the same instruction in the two syntaxes.

Most of the sixteen x86 condition codes have two synonyms (see Intel® SDM volume 1 appendix B). The use of any particular synonym in an example below is completely arbitrary and should not be interpreted as a requirement. Preferably, an assembler should be able to accept all synonyms.

All assembly code examples are given in lowercase but are in fact case-insensitive.

1 Extended GPRs

The 16 new GPRs (R16 to R31) do not require new syntax other than new register numbers. They and their subcomponents follow the same naming conventions as those for R8 to R15.

2 NDD (New Data Destination)

The NDD register is given as the first operand in the Intel syntax and the last operand in the AT&T syntax. The other operands keep their existing order in the respective syntax. Examples:

Intel syntax	**AT&T syntax**
`sub eax, dword ptr [rbx], ecx`	`subl %ecx, (%rbx), %eax`

The above SUB instruction subtracts the value of ECX from the 32b value loaded from the address in RBX, stores the result in EAX, and updates the architectural status flags accordingly.

3 ZU Indication

The only instructions that can use the ZU (zero-upper) indication are the SETcc instructions in EVEX map 4 and the IMUL instructions with opcodes 0x69 and 0x68 in EVEX map 4. The ZU indication is expressed as a suffix in the forms IMULZU and SETZUcc. (Note that “ZU” appears before the condition code in SETZUcc.) In the AT&T syntax, the “zu” should appear before the OSIZE modifier.

Intel syntax	**AT&T syntax**
`imulzu ax, word ptr [rbx], 0xab` `setzunz al`	`imulzuw $0xab, (%rbx), %ax` `setzuneb %al`

4 NF Indication

The following instruction classes can have the NF indication:

INC, DEC, NEG, ADD, SUB, AND, OR, XOR, SAL, SAR, SHL, SHR, ROL, ROR, SHLD, SHRD,
IMUL, IDIV, MUL, DIV, LZCNT, TZCNT, POPCNT,
ANDN, BEXTR, BLSI, BLSMSK, BLSR, BZHI

4.1 The {nf} Pseudo Prefix

The preferred syntax for expressing the NF indication is to use an “nf” pseudo prefix.

Intel syntax

```
{nf} add word ptr [rbx], cx
{nf} sub eax, dword ptr [rbx], ecx
{nf} imul ax, word ptr [rbx], 0xab
{nf} imulzu ax, word ptr [rbx], 0xab
```

AT&T syntax

```
{nf} addw %cx, (%rbx)
{nf} subl %ecx, (%rbx), %eax
{nf} imulw $0xab, (%rbx), %ax
{nf} imulzuw $0xab, (%rbx), %ax
```

4.2 The “nf” Suffix

If the assembler does not support pseudo prefixes, then the NF indication can be expressed by adding an “nf” suffix to the instruction’s mnemonic. In the AT&T syntax, the “nf” should appear before the OSIZE modifier.

For the IMUL instructions with opcodes 0x69 and 0x68 in EVEX map 4, it is possible that both “nf” and “zu” suffixes are present. In this case, “NF” should come before “zu”.

Intel syntax

```
addnf word ptr [rbx], cx
subnf eax, dword ptr [rbx], ecx
imulnf ax, word ptr [rbx], 0xab
imulnfzu ax, word ptr [rbx], 0xab
```

AT&T syntax

```
addnfw %cx, (%rbx)
subnfl %ecx, (%rbx), %eax
imulnfw $0xab, (%rbx), %ax
imulnfzuw $0xab, (%rbx), %ax
```

5 PUSH2 and POP2

In general, the Intel syntax and the AT&T syntax have opposite operand orders. In order to be consistent with this convention, the push and pop orders in (respectively) PUSH2 and POP2 instructions are read from left to right in the Intel syntax and right to left in the AT&T syntax.

In the examples below, the PUSH2 instruction pushes RAX and then RBX onto the stack, the POP2 instruction pops the stack to RBX and then to RAX.

Intel syntax	AT&T syntax
`push2 rax, rbx`	`push2q %rbx, %rax`
`pop2 rbx, rax`	`pop2q %rax, %rbx`

6 PPX

The PPX (Push Pop Xcelleration) hint is given as a “p” suffix of the instruction mnemonic. The OSIZE modifier “q” in the AT&T syntax may be omitted, because PPX is supported only for the push and pop of 64b registers.

Intel syntax	AT&T syntax
`pushp rax`	`pushpq %rax`
`pop2p rax, rbx`	`pop2pq %rax, %rbx`

7 CCMP and CTEST

The source condition code (SCC) is given as a suffix of CCMP or CTEST in the instruction mnemonic. In the AT&T syntax, the SCC should precede the OSIZE modifier. The names for the source condition codes are the same as those for the normal condition codes (see Intel® SDM volume 1 appendix B), except that the SCC values 0b1010 and 0b1011 (which are P and NP in normal condition codes) should be written as T and F because they now mean True and False, respectively.

The syntax recommendations for the DFV (default flags value) of CCMP and CTEST are given below.

7.1 The Finite Set Notation for DFV

This is the preferred syntax.

The OSZC flags setting is given as a list of flags in curly brackets and after the string “dfv=”, where only the flags being set to 1 are listed and any flag which is not listed is set to 0. Thus “`{dfv=}`” means that all of the OSZC flags are set to 0. The flag names within the curly bracket are separated by commas and their order should not matter. This notation is reminiscent of the notation for finite sets in mathematics, which should be familiar to almost all programmers.

Intel syntax	AT&T syntax
`ccmpnz {dfv=sf,cf,of} rax, qword ptr [rbx]`	`ccmpneq {dfv=sf,cf,of} (%rbx), %rax`
`ctestz {dfv=} ecx, ecx`	`ctestel {dfv=} %ecx, %ecx`

7.2 The Immediate Notation for DFV

If the assembler cannot support the finite set notation for DFV above, the following notation may be used. But it is less clear than the finite set notation and requires the programmer to remember the order of the flag bits.

The OSZC flags setting is given as an unsigned immediate in the range [0,...,15], corresponding to the value of [OF,SF,ZF,CF] viewed as a 4-bit unsigned integer. The previous two examples expressed in the immediate notation are shown below.

Intel syntax	**AT&T syntax**
`ccmpnz 13, rax, qword ptr [rbx]`	`ccmpneq $13, (%rbx), %rax,`
`ctestz 0, ecx, ecx`	`ctestel $0, %ecx, %ecx`

8 Explicit Prefix Selection

Whenever possible, the assembler should support the capability to explicitly select a REX2 or EVEX prefix for encoding an instruction when that prefix is not the default one. (Of course, it is assumed that the selected prefix is capable of supporting the intended semantics.) For example, the first instruction below has a REX prefix by default. The second and third instructions have the same semantics but are encoded using the REX2 and EVEX prefixes, respectively.

Intel syntax	**AT&T syntax**
`add rax, rbx`	`addq %rbx, %rax`
`{rex2} add rax, rbx`	`{rex2} addq %rbx, %rax`
`{evex} add rax, rbx`	`{evex} addq %rbx, %rax`

9 Change Log

Revision Number	Description	Date
1.0	1. Initial document release.	March 1, 2024