# Intel® Advanced Performance Extensions (Intel® APX)

## Architecture Specification

August, 2023
Revision 2.0

Document Number: 355828-002US

**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

# List of Figures

# List of Tables

# Chapter 1

# CHANGES

| Revision Number | Description | Date |
|---|---|---|
| 1.0 | 1. Initial document release | July 24, 2023 |
| 2.0 | 1. Updated encodings for CCMP/CTEST (all operations are encoded as ND=0, not ND=1)<br>2. Updated exception semantics for EVEX-promoted SSE-like instructions<br>3. Synchronized instruction table with per-instruction page listings | August 10, 2023 |

# Chapter 2

# CPUID

This section summarizes the CPUID names and leaf mappings referenced in this document.

| CPUID | Allocation |
|---|---|
| ADX | CPUID.(0x7.0x0).EBX[19] |
| AMX-TILE | CPUID.(0x7.0x0).EDX[24] |
| APX_F | CPUID.(0x7.0x1).EDX[21] |
| AVX10.1 | CPUID.(0x7.0x1).EDX[19] and CPUID.(0x24.0x0).EBX[7:0] >= 1 |
| AVX512BW | CPUID.(0x7.0x0).EBX[30] |
| AVX512DQ | CPUID.(0x7.0x0).EBX[17] |
| AVX512F | CPUID.(0x7.0x0).EBX[16] |
| BMI1 | CPUID.(0x7.0x0).EBX[3] |
| BMI2 | CPUID.(0x7.0x0).EBX[8] |
| CET | CPUID.(0x7.0x0).ECX[7] |
| CMPCCXADD | CPUID.(0x7.0x1).EAX[7] |
| ENQCMD | CPUID.(0x7.0x0).ECX[29] |
| INVPCID | CPUID.(0x7.0x0).EBX[10] |
| KEYLOCKER | CPUID.(0x7.0x0).ECX[23] |
| KEYLOCKER_WIDE | CPUID.(0x7.0x0).ECX[23] and CPUID(0x19.0x0).EBX[0] and CPUID(0x19.0x0).EBX[2] |
| LZCNT | CPUID.(0x80000001.None).ECX[5] |
| MOVBE | CPUID.(0x1.None).ECX[22] |
| MOVDIR | CPUID.(0x7.0x0).ECX[28] |
| SHA | CPUID.(0x7.0x0).EBX[29] |
| VMX | CPUID.(0x1.None).ECX[5] |

# Chapter 3

# INTRODUCTION

# 3.1 INTEL® APX INTRODUCTION

## 3.1.1 Introduction

Intel® Advanced Performance Extensions (Intel® APX) expands the Intel® 64 instruction set architecture with access to more registers and adds various new features that improve general-purpose performance. The extensions are designed to provide efficient performance gains across a variety of workloads without significantly increasing silicon area or power consumption of the core.

The main features of Intel® APX include:

- 16 additional general-purpose registers (GPRs) R16–R31, also referred to as Extended GPRs (EGPRs) in this document;
- Three-operand instruction formats with a new data destination (NDD) register for many integer instructions;
- Conditional ISA improvements: New conditional load, store and compare instructions, combined with an option for the compiler to suppress the status flags writes of common instructions;
- Optimized register state save/restore operations;
- A new 64-bit absolute direct jump instruction.

This introduction has two parts. The first part is an overview of Intel® APX instructions and their encoding formats. The second part describes the overall system architecture of Intel® APX and how it co-exists with existing x86 features.

In this document we will use the following abbreviations:

- “SDM” stands for Intel® 64 and IA-32 Architectures Software Developer Manuals.
- “OSIZE” stands for operand size.
- “ASIZE” stands for address size.

## 3.1.2 Intel® APX Instruction Format

This chapter details the encoding format of Intel® APX instructions. Intel® APX introduces a new 2-byte REX2 prefix (Section 3.1.2.1) and extends the existing 4-byte EVEX prefix (Section 3.1.2.3) in order to support 32 general-purpose registers (GPRs), the new data destination (NDD) register, and other enhancements.

The REX2 prefix and the extended EVEX prefix are available only after Intel® APX is enabled in 64-bit mode (see Section 3.1.4.2.1).

In general, x86 instructions have three separate encoding spaces: legacy, VEX, and EVEX. Instructions in the VEX and EVEX spaces are encoded using (respectively) the VEX prefixes (0xC4 and 0xC5) and the EVEX prefix (0x62). The legacy space consists of instructions which are not in either VEX or EVEX space. Each encoding space has a number of separate maps. Currently the legacy space has four maps numbered 0, 1, 2 and 3, which correspond to 1-byte opcodes (no escape), 2-byte opcodes (escape 0x0F), and 3-byte opcodes (escapes 0x0F38 and 0x0F3A), respectively. The VEX and EVEX spaces do not use escape bytes but encode the map id in the VEX or EVEX payload. The VEX and EVEX prefixes can support up to 32 and 8 maps, respectively.

The following is an overview of how Intel® APX impacts these three encoding spaces, the details of which are given in the subsequent sections:

- Legacy space:
  - All instructions in legacy maps 0 and 1 that have explicit GPR or memory operands can use the REX2 prefix to access the upper 16 GPRs (namely, R16 to R31).
    - There is one exception concerning XSAVE*/XRSTOR* for system architecture reasons.
  - Certain rows of opcodes in legacy maps 0 and 1 which do not have explicit GPR or memory operands are reserved under REX2 for future use.
    - One of these opcodes is already used by Intel® APX to encode a 64-bit absolute direct jump.
  - Select instructions from all four legacy maps are *promoted* into the EVEX space to enable the new capabilities provided by Intel® APX.
  - Instructions in legacy maps 2 and 3 cannot use the REX2 prefix and hence cannot access the upper 16 GPRs directly. But some of them have promoted forms (see the last item) that can.
- VEX space:
  - Select instructions from the VEX space are promoted into the EVEX space to enable the new capabilities provided by Intel® APX.
  - Otherwise, instructions in the VEX space cannot use the new capabilities provided by Intel® APX directly.
- EVEX space:
  - All instructions in the EVEX space can access the upper 16 GPRs in their memory operands.
  - All instructions promoted from the legacy space are placed in a single map, map 4, which was previously reserved.
  - Instructions promoted from the VEX space keep their previous map numbers, but with new EVEX forms added.

#### 3.1.2.1 REX2 Prefix

As shown in Figure 3.1, the REX2 prefix is two bytes long and consists of a byte of value 0xD5 followed by a second byte containing payload bits. The payload bits [W,R3,X3,B3] have the same meanings as the REX payload bits [W,R,X,B], except that the instructions “PUSH reg” with opcodes 0x50-0x57 and “POP reg”

| REX2 | Payload byte 1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0xD5 | M0 | R4 | X4 | B4 | W | R3 | X3 | B3 |

Figure 3.1: REX2 prefix

with opcodes 0x58-0x5F in legacy map 0 will use REX2.W to encode the PPX (push-pop acceleration) hint (see Section 3.1.3.1.2 for details). The payload bits [R4,X4,B4] provides the fifth and most significant bits of the R, X and B register identifiers, each of which can now address all 32 GPRs. Like the REX prefix, when OSIZE = 8b, the presence of the REX2 prefix makes GPR ids [4,5,6,7] address byte registers [SPL,BPL,SIL,DIL], instead of [AH,CH,DH,BH].

A REX2-prefixed instruction is always interpreted as an instruction in legacy map 0 or 1, with REX2.M0 encoding the map id. REX2 does not support any instruction in legacy maps 2 and 3. Intel® APX extension of legacy instructions in maps 2 and 3 (such as CRC32 and SHA instructions) is provided by the extended EVEX prefix (see Section 3.1.2.3.1).

REX2 is applicable to all instructions in maps 0 and 1 of the legacy space except the following:

- Prefixing XSAVE* and XRSTOR* instructions with REX2 triggers #UD. This is because XSAVE* and XRSTOR* are not allowed to use the upper 16 GPRs for system architecture reasons explained in Section 3.1.4.1.2.
- All opcodes listed below are reserved under REX2 and triggers #UD when prefixed with REX2:
  - Legacy map 0:
    - All opcodes in the row 0x4*
    - All opcodes in the row 0x7*
    - All opcodes in the row 0xA*
      - Exception: 0xA1 prefixed by REX2 is used to encode the JMPABS instruction (Section 3.1.3.3)
    - All opcodes in the row 0xE*
  - Legacy map 1:
    - All opcodes in row 0x3*
    - All opcodes in row 0x8*

  None of the above opcode encodes an instruction that needs an R, X or B register id and hence has no use for the REX2 prefix.

Any opcode in legacy map 0 or 1 that already #UD without REX2 will continue to #UD if prefixed by REX2. Furthermore, since the byte following a REX2 prefix is always interpreted as the main opcode byte, any legacy prefix byte (namely, 0x66, 0x67, 0xF0, 0xF2, 0xF3, and segment overrides) or a REX prefix byte (0x4*) following a REX2 prefix with REX2.M0 = 0 must #UD, because none of those bytes is the opcode of a valid instruction in legacy map 0 in 64-bit mode.

Note that the R, X and B register identifiers can also address non-GPR register types, such as vector registers, control registers and debug registers. When any of them does, the highest-order bits REX2.R4,

REX2.X4 or REX2.B4 are generally ignored, except when the register being addressed is a control or debug register. For example, using the REX2 prefix, the instruction “ADDPD xmm1, xmm2/m128” can use all 32 GPRs as the base and/or index registers in its memory operand, but it still cannot access XMM16 to XMM31 because REX2.R4 and REX2.B4 are ignored when the R and B register identifiers address vector registers. Similarly, when there is no index register, REX2.X4 and REX2.X3 are both ignored and code-generators should set these bits to zero.

The exception is that REX2.R4 and REX2.R3 are *not* ignored when the R register identifier addresses a control or debug register. Furthermore, if any attempt is made to access a non-existent control register (CR*) or debug register (DR*) using the REX2 prefix and one of the following instructions:

“MOV CR*, r64”, “MOV r64, CR*”, “MOV DR*, r64”, “MOV r64, DR*”.

#UD is raised.

| Encoding | Usage/Meaning | Prefix before REX2 |
|---|---|---|
| 0x2E | CS (NOTTAKEN-HINT) | Legal |
| 0x36 | SS | Legal |
| 0x3E | DS (CET-NOTRACK, and TAKEN-HINT) | Legal |
| 0x26 | ES | Legal |
| 0x4* | REX | Illegal |
| 0x62 | EVEX | Impossible |
| 0x64 | FS | Legal |
| 0x65 | GS | Legal |
| 0x66 | OSIZE | Legal |
| 0x67 | ASIZE | Legal |
| 0xC4 | VEX3 | Impossible |
| 0xC5 | VEX2 | Impossible |
| 0xD5 | REX2 | Impossible |
| 0xF0 | LOCK | Legal |
| 0xF2 | REPNE | Legal |
| 0xF3 | REPE | Legal |

Table 3.1: Legacy Prefix Applicability with REX2

REX2 must be the last prefix. The byte following it is interpreted as the main opcode byte in the opcode map indicated by M0. The 0x0F escape byte is neither needed nor allowed. (That is, the REX2 prefix followed by 0x0F triggers #UD.) The prefixes which may precede the REX2 prefix are LOCK (0xF0), REPE (0xF3), REPNE (0xF2), OSIZE override (0x66), ASIZE override (0x67), and segment overrides, all of which keep their current meanings and restrictions. (For example, a REX2-prefixed ADD whose destination is not a memory operand must #UD if it has a LOCK prefix.) A REX prefix (0x4*) immediately preceding the REX2 prefix is not allowed and triggers #UD. It is impossible for an EVEX (0x62), VEX2 (0xC5), VEX3 (0xC4), or

another REX2 to precede a REX2 prefix, because the first byte following any of these prefixes is interpreted as the main opcode byte. The prefix rules for REX2 are summarized in Table 3.1.

#### 3.1.2.2 New Data Destination

In a typical x86 integer instruction, the destination register or memory operand is also the first source operand. Intel® APX extends many such instructions with a new form that has an extra register operand called a **new data destination** (NDD). In such forms, NDD is the new destination register receiving the result of the computation and all other operands (including the original destination operand) become read-only source operands. This feature is illustrated in Table 3.2 using a typical 1-source operation (INC) and a typical 2-source operation (SUB). The NDD form keeps the same source operand order and encoding as the existing x86 form from which it is derived, but is placed in the EVEX space with the V register identifier encoding the NDD register (see Section 3.1.2.3.1). Note that this is a different use of the V register identifier from that in Intel® AVX and Intel® AVX-512 instructions, where V is typically used to encode a non-destructive source (NDS).

The NDD form does not change how the operation of the instruction updates the status flags, except when status flags update is explicitly suppressed by EVEX.NF = 1 (see section 3.1.2.3.1).

Unlike the merge-upper behavior at a destination register of a typical x86 integer instruction when OSIZE is 8b or 16b, the NDD register is always zero-uppered (see Section 3.1.2.4).

| Existing x86 form | Existing x86 semantics | NDD extension | NDD semantics |
|---|---|---|---|
| INC r/m | r/m := r/m + 1 | INC ndd, r/m | ndd := r/m + 1 |
| SUB r/m, imm | r/m := r/m - imm | SUB ndd, r/m, imm | ndd(v) := r/m - imm |
| SUB r/m, reg | r/m := r/m - reg | SUB ndd, r/m, reg | ndd(v) := r/m - reg |
| SUB reg, r/m | reg := reg - r/m | SUB ndd, reg, r/m | ndd(v) := reg - r/m |

Table 3.2: NDD Extensions of Typical Integer Instruction Forms

#### 3.1.2.3 Extended EVEX Prefix

The extended EVEX prefix is based on the current 4-byte EVEX prefix with the semantics of several payload bits re-defined. It is used to provide Intel® APX features for legacy instructions that cannot be provided by the REX2 prefix (such as the new data destination) and Intel® APX extensions of VEX and EVEX instructions. The payload bits which are shared by all uses of the extended EVEX prefix are shown in Figure 3.2. Most bits in the third payload byte (except for the V4 bit) are left unspecified in Figure 3.2 because the payload bit assignment depends on whether the EVEX prefix is used to provide Intel® APX extension to a legacy, VEX, or EVEX instruction, the details of which will be given in the subsections below.

The byte following the extended EVEX prefix is always interpreted as the main opcode byte. Escape

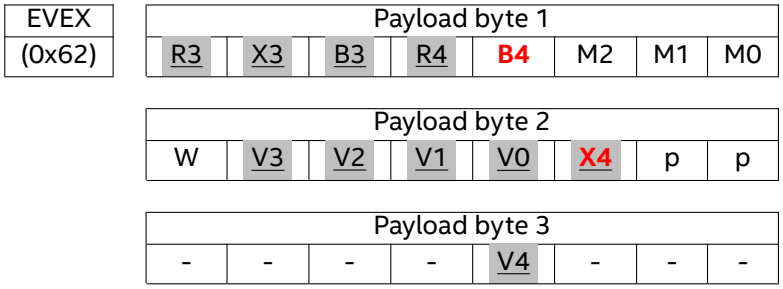


Figure 3.2: Extended EVEX prefix - Extensions for EGPRs only

sequences 0x0F, 0x0F38 and 0x0F3A are neither needed nor allowed. The map id of the instruction is encoded by the three bits [M2,M1,M0]. Thus the extended EVEX prefix can access up to 8 opcode maps.

The underlined bit fields (such as R3) are inverted. (They also have a light gray background.) The two bits shown in red boldface font are repurposed reserved bits used to provide the fifth and most significant bits of the B and X register identifiers. Their polarities are chosen so that the current fixed values at those two positions encode logical 0 after the repurposing. (In other words, the current fixed value at B4 is 0 and that at X4 is 1.)

The prefix rules for the extended EVEX prefix are the same as for the current EVEX prefix. The extended EVEX prefix must be the last prefix preceding the main opcode byte. The only prefixes which may precede the extended EVEX prefix are ASIZE override (0x67) and segment overrides. The presence of any other prefix triggers #UD.

The extended EVEX prefix provides Intel® APX extension of a legacy or VEX instruction by **promoting** it into the EVEX space, meaning that one or more new instructions with the same or related instruction forms are added to the EVEX space. When a VEX instruction is promoted, neither its map id nor its opcode nor its instruction form is changed, the only purpose of the promotion being to enable the instruction to access the extended GPRs and (for some instructions) to suppress a status flags update. For a legacy instruction, the notion of promotion is more complex. In addition to enabling it to access the extended GPRs, a legacy instruction may be promoted to support an NDD (new data destination) or ZU (zero-upper) form, to suppress status flags update, or even to express a related but quite different semantics. The various reasons for legacy instruction promotion are discussed in Section 3.1.2.3.1. All promoted legacy instructions are placed in a single map, EVEX map 4, which was previously reserved. Most promoted legacy instructions keep their previous opcodes, but not always. The general rules are documented in Section 3.1.2.3.1.

When a promoted legacy or VEX instruction has a memory operand with an 8b displacement (disp8), its scaling factor N is always 1. For existing EVEX instructions, Intel® APX does not change the existing disp8 scaling behaviors. (This notion is explained in SDM, vol.2, sec.2.7.5, “Compressed Displacement (disp8*N) Support in EVEX”.)

The EVEX-promoted operations of Intel® APX have different exception semantics compared with existing EVEX exception classes. These differences are similar in the way that VEX-encoded BMI instructions have

different exception semantics compared to "regular" VEX instructions. The differences in behavior include:

- Legacy-promoted Intel® APX EVEX instructions that rely solely on GPRs will not have CR0.TS sensitivity, and will not raise #NM exceptions
- VEX-promoted Intel® APX EVEX instructions that rely solely on GPRs will not have CR0.TS sensitivity, and will not raise #NM exceptions

A complete list of EVEX-promoted Intel® APX instructions can be found in Chapter 3.1.5.

| EVEX | Payload byte 1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| (0x62) | R3 | X3 | B3 | R4 | **B4** | 1 | 0 | 0 |

| Payload byte 2 | | | | | | | |
|---|---|---|---|---|---|---|---|
| W | V3 | V2 | V1 | V0 | **X4** | p | p |

| Payload byte 3 | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | **ND** | V4 | **NF** | 0 | 0 |

Figure 3.3: EVEX extension of legacy instructions

**3.1.2.3.1 EVEX Extension of Legacy Instructions** Figure 3.3 shows the payload bit assignment for the extended EVEX prefix when it is used to promote a legacy instruction into the EVEX space. As already mentioned, all those instructions are placed in EVEX map 4.

The W and pp bits have their current meanings in the EVEX prefix, except that pp = 0b01 can also be interpreted as OSIZE override for promoted integer instructions that have variable OSIZE. (But note that placing an explicit OSIZE override prefix 0x66 before the extended EVEX prefix triggers #UD.) Like the relationship between REX.W/REX2.W and the 0x66 prefix, EVEX.W = 1 takes precedence over EVEX.pp = 0b01 and makes OSIZE = 64b for instructions that have variable OSIZE.

The meanings of the ND and NF bits will be explained later. If any of the bits marked as 0 in the last payload byte is set to 1, #UD is raised, except for the CCMP and CTEST instructions (see Section 3.1.3.2.1), which uses two of the zero bits. There are further requirements on when the ND or NF bit must be set to 0, which will be given later.

For EVEX map 4, when OSIZE = 8b, GPR register ids [4,5,6,7] address byte registers [SPL,BPL,SIL,DIL], instead of [AH,CH,DH,BH].

As in REX2, when any of the bits in EVEX.{R4,X4,B4,R3,X3,B3} are not used by a promoted legacy instruction, it is ignored, and code-generators should set these bits to their logical 0 value (i.e., 1 for inverted bit fields, 0 for regular bit fields).

When an instruction can be encoded using either REX2 or EVEX prefix, the REX2 encoding is naturally to be preferred because it is two bytes shorter.

Note that it is possible for an EVEX-encoded legacy instruction to reach the 15-byte instruction length limit: 4 bytes of EVEX prefix + 1 byte of opcode + 1 byte of ModRM + 1 byte of SIB + 4 bytes of displacement + 4 bytes of immediate = 15 bytes in total. In such a case, no additional (ASIZE or segment override) prefix can be used. Since this limit is reached only when there is a long immediate, software can first load the immediate into a register and then apply the desired prefix(es) to the shorter register-source version of the same instruction class.

**Choice of Legacy Instructions to Promote** The set of legacy instructions that Intel® APX promotes into the EVEX space are chosen according to the following rules:

1. The following legacy instructions are *not* promoted:
   (a) LOCK-prefixed instructions.
   (b) String and input/output instructions, NOPs, UDs, PREFETCH*, PCLMULQDQ, XLAT, XSAVE* and XRSTOR*.
   (c) x87, MMX, SSE, MPX, GFNI, AES instructions.
      - Except for POPCNT and CRC32, which are promoted.
   (d) Any instruction which does not have *explicit* GPR or memory operands.
      - Example: ADD with opcode 0x05.
2. Among the remaining legacy instructions, the following ones are promoted:
   (a) Instructions that support NDD (new data destination):[1]

      INC, DEC, NOT, NEG, ADD, SUB, ADC, SBB, AND, OR, XOR, SAL, SAR, SHL, SHR, RCL, RCR, ROL, ROR, SHLD, SHRD, ADCX, ADOX, CMOVcc, and IMUL with opcode 0xAF in map 1

   For these instructions, EVEX.ND may be either 0 or 1. If EVEX.ND = 0, there is no NDD and EVEX.[V4,V3,V2,V1,V0] must be all zero. On the other hand, if EVEX.ND = 1, there is an NDD whose register id is encoded by EVEX.[V4,V3,V2,V1,V0]. The NDD and non-NDD versions of an instruction are related in the manner shown in Table 3.2.
      - *Note:* EVEX.[V4,V3,V2,V1,V0] must be set to zero for all promoted legacy instructions which are not in the above list and are not PUSH2 or POP2 (see Section 3.1.3.1.1) or CCMP or CTEST (see Section 3.1.3.2.1).
   (b) Instructions that support ZU (zero upper):

      IMUL with opcodes 0x69 and 0x6B in map 0 and SETcc instructions

   Although these instructions do not support NDD, the EVEX.ND bit is used to control whether its destination register has its upper bits (namely, bits [63:OSIZE]) zeroed when OSIZE is 8b or 16b. That is, if EVEX.ND = 1, the upper bits are always zeroed; otherwise, they keep the old values when OSIZE is 8b or 16b. For these instructions, EVEX.[V4,V3,V2,V1,V0] must be all zero.
      - *Note:* The notion of ZU does *not* apply to a memory destination: "SETcc mem" always writes a single byte of memory regardless of the value of EVEX.ND.

[1] For each operand type combination, the mnemonics SAL and SHL have the same semantics. But since both are mentioned in SDM, both are listed here as well. Note also that for each operand type combination, there are two opcodes for SAL/SHL. For example, both "0xD4 /4" and "0xD4 /6" encode "SAL/SHL r/m8, 1" and have the same semantics.

- *Note:* EVEX.ND must be set to zero for all promoted legacy instructions which do not support NDD or ZU and are not PUSH2 or POP2 (see Section 3.1.3.1.1) or CCMP or CTEST (see Section 3.1.3.2.1) or CMOVcc or CFCMOVcc (see Section 3.1.3.2.2).

(c) Instructions that support NF (status flags update suppression, hence "no flags"):

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

For these instructions, setting EVEX.NF = 1 suppresses the update of status flags while setting EVEX.NF = 0 keeps the current flags update behavior. For instructions that support both NDD and NF, the two features operate orthogonally with respect to each other.

- *Note:* EVEX.NF has special interpretations in PUSH2 and POP2 (see Section 3.1.3.1.2) and CMOVcc and CFCMOVcc (see Section 3.1.3.2.2) and does not mean "no flags" in them.
- *Note:* EVEX.NF must be set to zero in any promoted legacy instruction that is not in the above list and is not PUSH2 or POP2 (see Section 3.1.3.1.2) or CMOVcc or CFCMOVcc (see Section 3.1.3.2.2).

(d) The following instructions are also promoted into the EVEX space:

CMP, TEST, PUSH with opcode 0xFF and POP with opcode 0x8F in map 0

But the EVEX versions of these instructions have very different semantics from their legacy versions and will be given different mnemonics. The details are explained in Chapter 3.1.3.

- *Note:* For PUSH with opcode 0xFF and POP with opcode 0x8F in map 0, only the register forms (namely, the ModRM.Mod = 3 case) of the instructions are promoted.

(e) All remaining instructions in legacy maps 2 and 3 are promoted into the EVEX space, so that their GPR and memory operands can access all 32 GPRs. None of these instructions supports ND or NF, so both bits plus EVEX.[V4,V3,V2,V1,V0] must all be set to zero.

- *Note:* The promoted versions of MOVBE will be extended to include the "MOVBE reg1, reg2" form (namely, the ModRM.Mod = 3 case) for both opcodes 0xF0 and 0xF1. This extension makes the promotion of BSWAP for NDD support unnecessary.
- *Note:* Some instructions promoted from legacy maps 2 and 3 can have XMM operands:

  AESDEC128KL, AESDEC256KL, AESENC128KL, AESENC256KL,
  SHA1MSG1, SHA1MSG2, SHA1NEXTE, SHA1RNDS4,
  SHA256MSG1, SHA256MSG2, SHA256RNDS2

  These promoted instructions still can access only the lower 16 XMM registers (XMM0 to XMM15) and any attempt to access the upper 16 XMM registers (XMM16 to XMM31) will trigger #UD. Furthermore, they retain their SSE-like behavior in *not* zeroing the YMM[256:128] and ZMM[512:128] parts of their destination vector register.

**Opcode Assignment of Promoted Legacy Instructions** When a legacy instruction is promoted to EVEX map 4, its opcode may or may not change. Here the "opcode" includes not only the main opcode byte, but also the ModRM.Reg extension (if it is used) and the mandatory prefix EVEX.pp. The detailed mapping from the old opcode to the new one is documented in Chapter 3.1.5. The general rules we followed in the opcode assignment are discussed below.

The first rule is that every instruction in the new EVEX map 4 has a ModRM byte.

The second rule is that all instructions promoted from legacy map 0 retain their current opcodes.

The third rule is that an instruction that has variable OSIZE needs to consume two EVEX.pp values (66 and NP), because we need to use EVEX.pp = 66 to encode the OSIZE override. Thus each such instruction will preclude the use of EVEX.pp = 66 to encode a different instruction. Furthermore, those instructions whose current opcode includes a mandatory F2 or F3 prefix but which have variable OSIZE (namely, {CRC32, POPCNT, LZCNT, TZCNT}) must be given new opcodes, because EVEX.pp cannot encode double prefixes 66+F2 or 66+F3.

On the other hand, if an instruction does not have variable OSIZE, then it can share the same main opcode byte with one that does by having a mandatory prefix F2 or F3. We take advantage of this by placing the SETcc instructions in the same row (row 4) as the four variants of CMOVcc and CFCMOVcc instructions described in Section 3.1.3.2.2. This makes all promoted instructions whose opcode byte contains a condition code to be placed in a single row, in which all instructions sharing the same main opcode byte also share the same condition code.

| EVEX |
|---|
| (0x62) |

| Payload byte 1 | | | | | | | |
|---|---|---|---|---|---|---|---|
| R3 | X3 | B3 | R4 | **B4** | M2 | M1 | M0 |

| Payload byte 2 | | | | | | | |
|---|---|---|---|---|---|---|---|
| W | V3 | V2 | V1 | V0 | **X4** | p | p |

| Payload byte 3 | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | L | 0 | V4 | **NF** | 0 | 0 |

Figure 3.4: EVEX extension of VEX instructions

**3.1.2.3.2 EVEX Extension of VEX Instructions** Figure 3.4 shows the payload bit assignment for the extended EVEX prefix when it is used to promote a VEX instruction into the EVEX space.

Currently only KMOV*, BMI, and several other families of VEX instructions are promoted into the EVEX space. (The precise list of promoted VEX instructions can be found in Chapter 3.1.5.) The NF bit is used to optionally suppress status flags update in the following BMI instructions:

ANDN, BEXTR, BLSI, BLSMSK, BLSR, BZHI

For all other promoted VEX instructions, NF must be set to 0. The B4 and X4 provides the fifth and most significant bits of the B and X register identifiers only when they are used to address GPRs; otherwise they are ignored. Other bit fields have the same meanings as in the VEX prefix. If any of the 0 bits in Figure 3.4 is set to 1, #UD must be raised.

Promoting a VEX instruction into the EVEX space does not change the map id, the opcode, or the operand encoding of the VEX instruction.

An important point to note is that Intel® APX does *not* promote VEX instructions operating on vector registers which do not already have EVEX counterparts, even when such an instruction has a memory

operand which can use GPRs as base and index registers. This point can be illustrated by the example of the AES instructions, four of which have both VEX and EVEX forms (AESDEC, AESDECLAST, AESENC, AESENCLAST) and two of which have only VEX forms (AESIMC, AESKEYGENASSIST). Only the former instructions can use all 32 GPRs and all 32 vector registers in their EVEX forms (see Section 3.1.2.3.3). The latter instructions can use only 16 GPRs and 16 vector registers because they do not currently have EVEX forms.

When any of the bits in EVEX.{R4,X4,B4} is not used by a promoted VEX instruction, it is ignored, and code-generators should set these bits to their logical 0 value (i.e., 1 for inverted bit fields, 0 for regular bit fields).

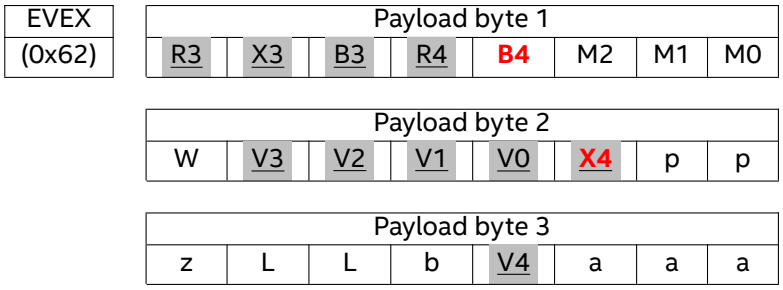


Figure 3.5: EVEX extension of EVEX instructions

**3.1.2.3.3 EVEX Extension of EVEX Instructions** All existing EVEX instructions are extended by Intel® APX using the extended EVEX prefix with payload bits shown in Figure 3.5, so that they can access all 32 GPRs. Except for the B4 and X4 bits, all other payload bits have the same meanings as they do now in the EVEX prefix.

| | 4 | 3 | [2:0] | Reg. Type | Common Usages |
|---|---|---|---|---|---|
| **REG** | EVEX.R4 | EVEX.R3 | modrm.reg | GPR, Vector | Destination or Source |
| **VVVV** | EVEX.V4 | EVEX.[V3,V2,V1,V0] | | GPR, Vector | Destination or Source |
| **RM (Vector)** | EVEX.X3 | EVEX.B3 | modrm.r/m | Vector | Destination or Source |
| **RM (GPR)** | EVEX.B4 | EVEX.B3 | modrm.r/m | GPR | Destination or Source |
| **BASE** | EVEX.B4 | EVEX.B3 | modrm.r/m | GPR | Memory addressing |
| **INDEX** | EVEX.X4 | EVEX.X3 | sib.index | GPR | Memory addressing |
| **VIDX** | EVEX.V4 | EVEX.X3 | sib.index | Vector | VSIB memory addressing |

Table 3.3: 32-Register Support in APX Using EVEX with Embedded REX Bits

Table 3.3 shows how Table 2-31, “32-Register Support in 64-bit Mode Using EVEX with Embedded REX Bits”, in SDM vol.2 sec.2.7.2, “Register Specifier Encoding and EVEX”, is to be adapted for APX. Note that the EVEX prefix already has provisions for extending the B and X register identifiers from 4 to 5 bits to address 32 vector registers, as described by the rows marked “RM (Vector) ” and “VIDX” in Table 3.3. For those purposes, the current scheme will continue to be used. The B4 and X4 bits are used only for the rows marked, “RM (GPR)”, “BASE”, and “INDEX” of Table 3.3. The most significant bits R4, B4, and X4 may be used to access the upper 16 GPRs (R16 to R31) only after APX has been enabled in 64-bit mode (see Section 3.1.4.2.1). Before APX is enabled, any attempt to access the upper 16 GPRs triggers #UD.

If any of the bits R4, B4, and X4 is not used by an EVEX instruction, it is ignored, and code-generators should set these bits to their logical 0 value (i.e., 1 for inverted bit fields, 0 for regular bit fields).

#### 3.1.2.4 Merge vs Zero-Upper at the Destination Register

The rules discussed in this section are applicable only when the destination of an instruction is a GPR. If the destination of an instruction is a memory location, the number of bytes being written to memory is always OSIZE/8 or zero.

Prior to Intel® APX, the following rules apply in 64-bit mode when an instruction’s destination is a GPR and OSIZE $<$ 64b:

1. If OSIZE is 32b, the destination GPR gets the instruction’s result in bits [31:0] and all zeros in bits [63:32].
2. If OSIZE is 8b or 16b, the destination GPR gets the instruction’s result in bits [OSIZE-1:0] but keep its old value in bits [63:OSIZE].

For an Intel® APX instruction, the above rules still apply when there is no NDD, namely, either when the REX2 prefix is used or when the EVEX prefix is used with EVEX.ND = 0.

For an Intel® APX instruction with an NDD (see items 2.(a) of Section 3.1.2.3.1), the destination GPR (namely, the NDD) will get the instruction’s result in bits [OSIZE-1:0] and, if OSIZE $<$ 64b, have its upper bits [63:OSIZE] zeroed. In other words, there is no merging of the old and new values at the NDD regardless of the OSIZE or whether the NDD is one of the source operands.

The ZU indication described in items 2.(b) of Section 3.1.2.3.1 does not introduce an NDD. For those instructions, EVEX.ND=0 keeps the current x86 behavior, but EVEX.ND=1 forces the zeroing of bits [63:OSIZE] for any OSIZE $<$ 64b.

CFCMOVcc (Conditionally Faulting CMOVcc) of the forms “CFCMOVcc reg, reg1” and “CFCMOVcc reg, mem” (see Section 3.1.3.2.2) follow the same rules as if reg were an NDD (namely, its bits [64:OSIZE] are zeroed). Additionally, if the condition code evaluate to false, reg is completely zeroed.

The NDD forms of CMOVcc and CFCMOVcc follow the general rules for NDD stated above.

### 3.1.3 Additional Intel® APX Instructions

#### 3.1.3.1 Register Save/Restore Optimizations

The addition of 16 GPRs can increase the number of GPR save/restore operations around procedure calls and returns. Thus Intel® APX provides two mechanisms for reducing the cost of pushing/popping GPRs to/from the stack.

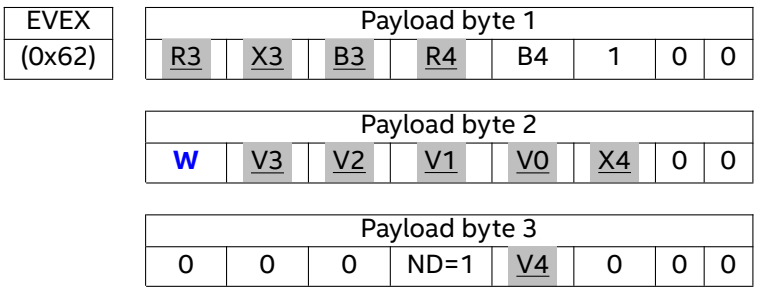


Figure 3.6: EVEX prefix for PUSH2 and POP2

**3.1.3.1.1 PUSH2 and POP2** PUSH2 and POP2 are two new instructions for (respectively) pushing/popping two GPRs at a time to/from the stack. PUSH2 and POP2 interpret the EVEX payload bits as shown in Figure 3.6. (The use of the W bit is explained in the next subsection.) The opcodes of PUSH2 and POP2 are those of "PUSH r/m" and "POP r/m" from legacy map 0, but we require ModRM.Mod = 3 in order to disallow a memory operand. (A PUSH2 or POP2 with ModRM.Mod $\neq$ 3 triggers #UD.) In addition, we require that EVEX.ND = 1, so that the V register identitifer is valid and specifies the additional register operand.

| Opcode | Instruction | Semantics |
|---|---|---|
| EVEX map=4 pp=0 ND=1 0xFF/6 Mod=3 | PUSH2 v64, b64 | PUSH v64<br>PUSH b64 |
| EVEX map=4 pp=0 ND=1 0x8F/0 Mod=3 | POP2 v64, b64 | POP v64<br>POP b64 |

Table 3.4: Summary of the encoding and semantics of PUSH2 and POP2

The encoding and semantics of PUSH2 and POP2 are summarized in Table 3.4, where b64 and v64 are the 64b GPRs encoded by the B and V register identifiers, respectively. (The OSIZE of PUSH2 and POP2 is always 64b.) The semantics is given in terms of an equivalent sequence of simpler instructions. We require further that neither b64 nor v64 be RSP and, for POP2, b64 and v64 be two different GPRs. Any violation

of these conditions triggers #UD. For PUSH2, the two register values being pushed are either both written to memory or neither one is written, but there is no guarantee that the two writes are performed together as a single atomic write.

The data being pushed/popped by PUSH2/POP2 must be 16B-aligned on the stack. Violating this requirement triggers #GP.

**3.1.3.1.2 Balanced PUSH/POP Hint** A PUSH and its corresponding POP may be marked with a 1-bit Push-Pop Acceleration (PPX) hint to indicate that the POP reads the value written by the PUSH from the stack. The processor tracks these marked instructions internally and fast-forwards register data between matching PUSH and POP instructions, without going through memory or through the training loop of the Fast Store Forwarding Predictor (FSFP).

When applying the PPX hint, the compiler needs to make sure that it always marks both the PUSH and its matching POP (i.e., the POP which reads from the same stack memory address that the PUSH writes to). This balancing rule naturally applies to PUSH/POP sequences in function prologs/epilogs, respectively. It does not apply to standalone PUSH sequences, such as function argument pushes onto the stack. Such sequences should not be marked with the PPX hint.

The PPX hint is encoded by setting REX2.W = 1 and is applicable only to PUSH with opcode 0x50+rd and POP with opcode 0x58+rd in the legacy space. It is not applicable to any other variants of PUSH and POP.

The PPX hint requires the use of the REX2 prefix, even when the functional semantics can be encoded using the REX prefix or no prefix at all. Note also that the PPX hint implies OSIZE = 64b and that it is impossible to encode PPX with OSIZE = 16b, because REX2.W takes precedence over the 0x66 prefix.

Similarly, PUSH2 can be marked with a PPX hint to indicate that it has a matching POP2, which is also marked. The PPX hint for PUSH2 and POP2 is encoded by setting EVEX.W = 1. We require that EVEX.pp = 0 in PUSH2 and POP2 and their OSIZE always be 64b.

Note that for PPX to work properly, a PPX-marked PUSH2 (respectively, POP2) should always be matched with a PPX-marked POP2 (PUSH2), not with two PPX-marked POPs (PUSHs).

The PPX hint is purely a performance hint. Instructions with this hint have the same functional semantics as those without. PPX hints set by the compiler that violate the balancing rule may turn off the PPX optimization, but they will not affect program semantics.

### 3.1.3.2 Conditional Instruction Set Extensions

The purpose of these instructions is to enable the compiler to more widely apply if-conversion to larger regions of code, while minimizing the risk of performance regressions if branches turn out to be well-predicted.

**3.1.3.2.1 Conditional CMP and TEST** CCMP and CTEST are two new sets of instructions for conditional CMP and TEST, respectively. They are encoded by promoting all opcodes of CMP and TEST, except for

those forms which do not have explicit GPR or memory operands, into the EVEX space and re-interpreting the EVEX payload bits as shown in Figure 3.7. Note that the V and NF bits and two of the zero bits are repurposed. The ND bit is required to be set to 0. There are no EVEX versions of CMP and TEST with EVEX.ND = 1.

| EVEX | Payload byte 1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| (0x62) | R3 | X3 | B3 | R4 | B4 | 1 | 0 | 0 |

| Payload byte 2 | | | | | | | |
|---|---|---|---|---|---|---|---|
| W | OF | SF | ZF | CF | X4 | p | p |

| Payload byte 3 | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | ND=0 | SC3 | SC2 | SC1 | SC0 |

Figure 3.7: EVEX prefix for conditional CMP and TEST

```
// CCMP

IF (src_flags satisfies scc):
    dst_flags = compare(src1,src2)
ELSE:
    dst_flags = flags(evex.[of,sf,zf,cf])
```

Figure 3.8: Pseudocode for CCMP

```
// CTEST

IF (src_flags satisfies scc):
    dst_flags = test(src1,src2)
ELSE:
    dst_flags = flags(evex.[of,sf,zf,cf])
```

Figure 3.9: Pseudocode for CTEST

The four SC* bits form a **source condition code** SCC = EVEX.[SC3,SC2,SC1,SC0], the encoding of which is the same as that of the existing x86 condition codes (SDM Volume 1, Appendix B, “EFLAGS Condition Codes”), with two exceptions:

- If SCC = 0b1010, then SCC evaluates to true regardless of the status flags value.
- If SCC = 0b1011, then SCC evaluates to false regardless of the status flags value.

Consequently, the SCC cannot test the parity flag PF.

The SCC is used as a predicate for controlling the conditional execution of the CCMP or CTEST instruction:

- If SCC evaluates to true on the status flags, then the CMP or TEST is executed and it updates the status flags normally. Note that the SCC = 0b1010 exception case can be used to encode unconditional CMP or TEST as a special case of CCMP or CTEST.
- If SCC evaluates to false on the status flags, then the CMP or TEST is not executed and instead the status flags are updated as follows:
  - OF = EVEX.OF
  - SF = EVEX.SF
  - ZF = EVEX.ZF
  - CF = EVEX.CF
  - PF = EVEX.CF
  - AF = 0

  Note that the SCC = 0b1011 exception case can be used to force any desired truth assignment to the flags [OF,SF,ZF,CF] unconditionally.

  Unlike the CMOVcc extensions discussed below, SCC evaluating to false does not suppress memory faults from a memory operand.

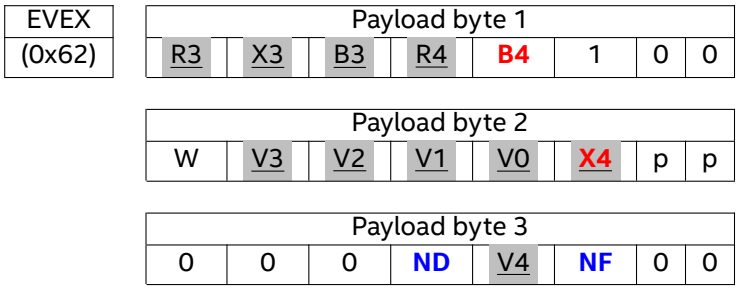


Figure 3.10: EVEX extension of CMOVcc instructions

**3.1.3.2.2 CMOVcc Extensions** There are four different forms of EVEX-promoted CMOVcc instructions (shown in Table 3.5) corresponding to the four possible combinations of the values of EVEX.ND and EVEX.NF (see Figure 3.10). Three of these forms have a new mnemonic, CFCMOVcc, where the "CF" prefix denotes "conditionally faulting" and means that all memory faults are suppressed when the condition code evaluates to false and the r/m operand is a memory operand. Note that EVEX.NF is used as a direction bit in the 2-operand case to reverse the source and destination operands.

If the destination of any of the four forms of CMOVcc and CFCMOVcc in Table 3.5 is a register, we require that the upper bits [63:OSIZE] of the destination register be zeroed whenever OSIZE $<$ 64b. But if the destination is a memory location, then either OSIZE bits are written or there is no write at all.

| EVEX.ND | EVEX.NF | Instruction Forms | Instruction Semantics |
|---|---|---|---|
| 0 | 0 | CFCMOVcc reg, r/m | IF (src_flags satisfies cc):<br>  reg := r/m<br>ELSE:<br>  // memory faults are suppressed<br>  reg := 0 |
| 0 | 1 | CFCMOVcc r/m, reg | IF (src_flags satisfies cc):<br>  r/m := reg<br>ELIF (r/m is a register):<br>  r/m := 0<br>ELSE:<br>  // memory faults are suppressed<br>  skip |
| 1 | 0 | CMOVcc ndd, reg, r/m | // memory faults are not suppressed<br>temp := r/m<br>IF (src_flags satisfies cc):<br>  ndd := temp<br>ELSE:<br>  ndd := reg |
| 1 | 1 | CFCMOVcc ndd, reg, r/m | IF (src_flags satisfies cc):<br>  ndd := r/m<br>ELSE:<br>  // memory faults are suppressed<br>  ndd := reg |

Table 3.5: New CMOVcc variants according to EVEX.ND and EVEX.NF controls

In contrast, the REX2 versions of CMOVcc have the same legacy behavior as the existing CMOVcc. In particular, the destination register is not zeroed and memory faults are not suppressed when the condition is false. This behavior keeps legacy CMOVcc operation semantics and timing in line with Intel's guidelines for mitigating timing side channels against cryptographic implementations.

**3.1.3.2.3 SETcc.zu** Intel® APX includes a new variant of SETcc, called SETcc.zu (zu = "zero upper"; see item 2(b) of Section 3.1.2.3.1). The semantics of "SETcc.zu dest" is shown in Figure 3.11.

Many existing SW usages of SETcc require pre-zeroing the register (often with a zero idiom) because of the partial register write semantics (merging with upper bits). SETcc.zu makes the pre-zeroing unnecessary.

```
IF (src_flags satisfies cc):
    dest[63:0] = 1
ELSE:
    dest[63:0] = 0;
```

Figure 3.11: Pseudocode for SETcc.zu

#### 3.1.3.3 64-bit Absolute Direct Jump

JMPABS transfers program control to the 64-bit absolute address target64 given as a quadword immediate. JMPABS is in legacy map 0 and requires a REX2 prefix with REX2.M0 = 0 and REX2.W = 0. All other REX2 payload bits are ignored, and code-generators should set these bits to 0. JMPABS does not have a ModRM byte and target64 is placed immediately after the opcode byte, so the entire instruction is 11 bytes long. Prefixing JMPABS with 0x66, 0x67, 0xF0, 0xF2, or 0xF3 triggers #UD. Segment overrides are allowed but ignored by JMPABS.

| Opcode | Instruction | Semantics |
|---|---|---|
| REX2 M0=0 W=0 0xA1 target64 | JMPABS target64 | Direct jump to absolute address target64 |

Table 3.6: Summary of the encoding and semantics of JMPABS

### 3.1.4 System Architecture

In total, Intel® APX includes:

1. New Intel® APX state (GPRs)
   (a) 16 additional GPRs (R16–R31), which are referred to as Extended GPRs (EGPRs).

2. Modified system state (existing state, but modified for Intel® APX)
   (a) CPUID Enumeration for APX_F (APX Foundation).
   (b) XCR0 Extensions.
   (c) XSAVE area for Intel® APX state.
3. Intel® APX prefixes
   (a) Two new prefixes (REX2 and Extended EVEX) that support EGPR addressing, new data destination (NDD), status flags update suppression, and a number of new instructions (see Sections 3.1.2 and 3.1.3 for details).

Intel® APX features are only available in IA-32e 64-bit Protected Mode, and are an XSAVE-enabled feature which requires XCR0 enabling before using the new Intel® APX ISA, new Intel® APX prefixes (REX2) and prefix extensions (EVEX extensions). See section 3.1.4.2 for details on XCR0-enabling for Intel® APX.

#### 3.1.4.1 New Intel® APX Register State

**3.1.4.1.1 Extended GPRs (EGPRs)** The new Extended GPRs (EGPRs) behave the same as legacy GPRs (R8-R15) from the perspective of RESET, liveness in non 64-bit modes, and architectural preservation in C6 and HDC (Hardware Duty Cycling). The main difference between EGPRs and legacy GPRs, is that EGPRs are not *enabled* by default (specifically XCR0-enabled) in 64-bit mode, and their INIT behavior is XMM-like; in that EGPRs and many other XSAVE-enabled register states are un-changed on INIT events as shown in Table 3.7.

| State | Power Up | RESET | INIT |
|---|---|---|---|
| R8-R15 | 0x0 | 0x0 | 0x0 |
| XMM0-XMM7 | 0x0 | 0x0 | Unchanged |
| XMM8-XMM15 | 0x0 | 0x0 | Unchanged |
| R16-R31 | 0x0 | 0x0 | Unchanged |

Table 3.7: Power-Up, Reset, INIT Behavior of EGPRs vs. Other Legacy State

**3.1.4.1.2 Extended GPR Access (Direct and Indirect)** The EGPRs are only directly accessible within 64-bit mode. Outside of 64-bit mode, the EGPRs can be indirectly accessed via XSAVE ISA features, as they are part of the Intel® APX extension to the user-level XSAVE area.

The EGPRs of Intel® APX, while only directly accessible in 64-bit mode, retain their values as this mode is entered/exited within the current execution context. Entering/leaving 64-bit mode via traditional (explicit) control flow does not directly alter the content of the EGPRs (EGPRs behave similar to R8-R15 in this

regard). Additionally, entering/leaving 64-bit mode via events, exceptions, interrupts, VM Exits, and system calls, does not directly alter the content of the EGPRs.

EGPR content is modified directly by Intel® APX instructions which choose to write EGPRs as destination registers, and indirectly via XRSTOR-like operations which target Intel® APX state through the use of a Requested Feature Bitmap (RFBM) with RFBM[APX_F]=1 (APX_F is index 19).

Intel® APX purposefully defines EGPRs as XSAVE-enabled state as a form of state encapsulation, which provides an easy path for Operating System (OS) and Virtual Machine Monitor (VMM) enabling of Intel® APX without necessitating that kernels/VMMs be re-compiled to use Intel® APX ISA themselves (i.e., does not require manually saving/restoring EGPRs using Intel® APX instructions). Furthermore, this also means that the Intel® APX enabling is more "portable" when it comes to co-existence with other x86 technologies that leverage XSAVE as part of their inner-workings (like Intel® SGX and Intel® TDX).

From an XSAVE perspective, EGPR state (R16-R31) are considered to be in INIT state if all of the registers have the value 0x0. XINUSE = 0 when this condition is met, although as with baseline x86 architecture, it's possible for all of the EGPRs to be 0x0, while XINUSE = 1. All instructions which can impact EGPR state (R16-R31), either directly or indirectly, are capable of toggling XINUSE trackers for EGPR state so that INIT/MODIFIED optimizations with respect to XSAVE occur properly. Intel® APX state can be made INIT only via the XRSTOR* instruction. No other instruction can put EGPRs into INIT state, at this time.

It is important to note that XSAVE/XRSTOR usage cannot use an EGPR as an operand. Any attempt to use an Intel® APX prefix with XSAVE/XRSTOR will #UD.

XSAVE/XRSTOR behavior for EGPRs has no modal specialization of behavior. As such, XSAVE/XRSTOR management of EGPRs outside of 64-bit mode will save/restore all EGPRs when requested.

#### 3.1.4.2 Modified System State

**3.1.4.2.1 CR and XCR Modifications** Intel® APX is an XSAVE-enabled feature, whose state can be managed using the suite of XSAVE/XRSTOR ISA, and whose state components can be enabled via alterations to XCR0/IA32_XSS. Intel® APX is enumerated as a single Intel® APX-enabled feature.

- New fields in XCR0:
  - APX_F – Intel® APX state and prefixes are governed by XCR0[APX_F=19]. This control bit enables Intel® APX ISA by enabling the use of the REX2 and Extended EVEX prefixes in IA-32e 64-bit mode and by enabling the XSAVE feature set to manage Intel® APX state. Note that in 64-bit mode, none of the Intel® APX features (including the REX2 and Extended EVEX prefixes and all new Intel® APX instructions) can be used until they are XCR0-enabled.

The #UD behavior for Intel® APX instructions are controlled by XCR0[APX_F] as shown in Table 3.8.

Where the determination of what classifies an APX instruction is:

- All REX2 prefixed instructions are considered APX instructions (regardless of register usage).

| CR4.OSXSAVE | XCR0[APX_F] | Response when executing an Intel® APX instruction |
|---|---|---|
| 0 | 0 | Fault (UD) - CR4.OSXSAVE gates XSAVE-enabled ISA usage |
| 0 | 1 | Fault (UD) - CR4.OSXSAVE gates XSAVE-enabled ISA usage, even when XCR0 bits are set |
| 1 | 0 | Fault (UD) - CR4.OSXSAVE is setup, but APX_F feature flag not enabled |
| 1 | 1 | Normal execution, subject to other, opcode-specific inherited CPUID/XCR0 rules |

Table 3.8: Intel® APX XCR0 and CR4 #UD Rules

- All Legacy and VEX instructions promoted into EVEX space are considered APX instructions (regardless of register/feature usage).
- All existing EVEX instructions which may use EVEX extensions are considered potential APX instructions. As such, EVEX payload fields retain their current meanings if APX is not enabled. In particular, EVEX.B4 and EVEX.X4 would remain reserved and would trigger an exception (UD, depending on XCR0[APX_F]) if either doesn't hold their current fixed ("reserved") values.

Important notes:

- XSAVE and XCR0 architecture treats CR4.OSXSAVE=1 as a pre-requisite for using XSAVE-enabled features. As such, when CR4.OSXSAVE=0, XCR0 is treated as all 0's. Therefore, when CR4.OSXSAVE=0, APX features are not available (as if XCR0[APX_F] was 0).
- APX-prefixed instructions and instructions which may use APX prefix payloads (REX2 and EVEX) may have legacy, or inherited, sensitivities. As an example, a vector instruction which chooses to use an EGPR is sensitive to both APX_F and Intel® AVX* CPUID and XCR0 requirements. Additionally, BMI instructions are sensitive to both APX_F and BMI* CPUID/XCR0 requirements.

#### 3.1.4.3 Intel® APX CPUID Enumeration and XSAVE Architecture

**3.1.4.3.1 Intel® APX Feature and Enumeration** Intel® APX is enumerated as a platform feature through the CPUID interface. Intel® APX features are available through the "Structured Extended Feature Flags Enumeration" CPUID interface, which is accessed via a new APX_F leaf of CPUID.(EAX=0x7, ECX=1).EDX[21] = 1

Intel® APX does not have an impact on IA32_CORE_CAPABILITIES, and IA32_ARCH_CAPABILITIES, and MSR_PLATFORM_INFO MSRs.

**3.1.4.3.2 Intel® APX Extended State Management** Intel® APX defines a single set of state that can be managed via XSAVE*/XRSTOR* instructions:

1. Intel® APX EGPR state (R16-R31) is save/restore controlled via XCR0[APX_F=19].

**Processor Extended State Enumeration Sub-leaf**

**User-level Intel® APX XSAVE area – CPUID.(EAX=0xD, ECX=19)**

- EAX (Size in bytes of XSAVE/XRSTOR area for this feature)
  - 128 (minimum size) derived from...
    - EGPR space = 8bytes*16registers = 128 bytes
- EBX (Offset in bytes in XSAVE/XRSTOR area for this feature)
  - 960 (0x3C0).
    - Intel® APX is feature index 19 in XCR0.
    - Intel® APX is architected to re-use the deprecated area of Intel® MPX.
- ECX (Controls for contiguity, XSS, and XFD controls)
  - 0x0 (0b000), derived from...
    - Where:
    - ECX[0] = 0 – alignment restriction (0 = no, 1 = yes)
    - ECX[1] = 0 – user-level/supervisor-level component (0 = user-XCR0, 1 = supervisor-XSS)
    - ECX[2] = 0 – XFD support (0 = no, 1 = yes)

#### 3.1.4.3.3 Intel® APX XSAVE Buffer Definition

**User-Level Intel® APX XSAVE Area Format** Historically, GPR state was not included in the XSAVE area. For Intel® APX the architecture purposefully encapsulates EGPRs as XSAVE-enabled state to provide options for state management without forcing software layers to explicitly use Intel® APX ISA. For instance, there may be systems where applications/guests make use of Intel® APX, but the supporting OS/VMM does not, and it is convenient to be able to save/restore Intel® APX EGPRs using XSAVE/XRSTOR (i.e., without manual save/restore). Hence, EGPRs are added to the user-level save area. By placing architectural Intel® APX state in the XSAVE area, the architecture makes it feasible for OS/VMMs to manage Intel® APX state on behalf of apps/guest without forcing OSs/VMMs to have manual, state-specific save/restore logic that may require kernel/VMM re-compilation with an Intel® APX-enabled compiler. In addition, inclusion as an XSAVE-enabled feature eases co-existence with features with XCR0-oriented interfaces, such as Intel® SGX and Intel® TDX.

**XSAVE Area Offset** The XSAVE footprint of Intel® APX, which re-uses (via re-definition) the 128B area of the now-deprecated Intel® Memory Protection Extensions (Intel® MPX). Since Intel® MPX had been previously deprecated, no processor will enumerate support for both Intel® MPX and Intel® APX. The architecture does not re-use any XCR0 control bits and instead only re-purposes the 128-byte XSAVE area that had been previously allocated by Intel® MPX (state component indices 3 and 4, making up a 128-byte area located at an offset of 960 bytes into an un-compacted XSAVE buffer). Intel® APX re-architects the

| Offset (in bytes) | Description | Width (in bytes) |
|---|---|---|
| 0 | EGPR-16 (APX, R16) | 8 |
| 8 | EGPR-17 (APX, R17) | 8 |
| 16 | EGPR-18 (APX, R18) | 8 |
| 24 | EGPR-19 (APX, R19) | 8 |
| 32 | EGPR-20 (APX, R20) | 8 |
| 40 | EGPR-21 (APX, R21) | 8 |
| 48 | EGPR-22 (APX, R22) | 8 |
| 56 | EGPR-23 (APX, R23) | 8 |
| 64 | EGPR-24 (APX, R24) | 8 |
| 72 | EGPR-25 (APX, R25) | 8 |
| 80 | EGPR-26 (APX, R26) | 8 |
| 88 | EGPR-27 (APX, R27) | 8 |
| 96 | EGPR-28 (APX, R28) | 8 |
| 104 | EGPR-29 (APX, R29) | 8 |
| 112 | EGPR-30 (APX, R30) | 8 |
| 120 | EGPR-31 (APX, R31) | 8 |

Table 3.9: XSAVE EGPR Layout

two previous 64-byte state components and uses them as a single state component housing 128-bytes of storage for EGPRs (8-bytes * 16 registers). Intel® APX uses XCR0 index 19, and as such, the monotonic relationship between an increasing XCR0 index and an increasing XSAVE buffer offset is altered. The logical ordering of the first 8 entries in the un-compacted XSAVE buffer with regards to XCR0 indices changes in the following manner:

- Before Intel® APX has been introduced:
  - 0, 1, 2, 3, 4, 5, 6, 7, ...
- After Intel® APX has been introduced:
  - 0, 1, 2, *19*, 5, 6, 7, ...

Conversely, in a compacted XSAVE buffer (via XSAVEC), which saves state components in a dynamic, XCR0 index-relative order, Intel® APX state would be placed later with respect to all state components with lesser XCR0 indices. Therefore, the logical order of Intel® APX state differs between un-compacted and compacted forms. Re-purposing the deprecated state area of Intel® MPX allows for Intel® APX to avoid potential interactions with being placed after large state components, such as Intel® AMX.

#### 3.1.4.4 Interactions with other IA Features

**3.1.4.4.1 VMX** Virtualization extensions operate essentially unchanged under Intel® APX, other than the architectural footprint of virtualization extensions expanding to Intel® APX state.

VMCS fields related to decoded instruction info are extended to support Intel® APX, namely:

- VM-Exit Instruction Information: A VMCS field that provides decoded instruction field info for certain types of exiting instructions, namely: CLTS, CPUID, ENCLS, GETSEC, HLT, IN, INS, INVD, INVEPT, INVLPG, INVPCID, INVVPID, LGDT, LIDT, LLDT, LMSW, LOADIWKEY, LTR, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, PCONFIG, RDMSR, RDPMC, RDRAND, RDSEED, RDTSC, RDTSCP, RSM, SGDT, SIDT, SLDT, STR, TPAUSE, UMWAIT, VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON, WBINVD, WBNOINVD, WRMSR, XRSTORS, XSETBV, and XSAVES (See Tables 28-8 through 28-15 of Volume 3 of the SDM).
- Exit qualifications for CR/DR access, namely MOV CR*, MOV DR*, LMSW, and CLTS (See Tables 28-3 and 28-4 in Volume 3 of the SDM)

These VMCS fields currently house 4-bit register IDs, and require architectural modifications to support EGPRs and their 5-bit register IDs. VM Exit qualification is extended in-place, while a new VMCS field is introduced to provide the extension for VM-Exit Instruction Information. This extension comes in the form of a new 64-bit field called the VM-Exit Extended Instruction-Information (EII) field. The field has space for a total of 4 register IDs (reg1, reg2, base, index) to match the current capabilities of all of the existing register fields in the VM-Exit Instruction-Information field.

The behavior of the aforementioned instructions which regards to Intel® APX features are shown in Table 3.10.

Any VM-exits which populated VM-Exit Instruction Info, along with instructions which populated exit qualification info with decoded information, will continue to populate the legacy fields in addition to the new VMCS field, called VM-Exit Extended Instruction Info. Architectural behaviors are as follows:

- Any instruction which has a defined VM-Exit Instruction Info field will populate *both* VM-Exit Instruction Info *and* VM-Exit Extended Instruction Info. The information in VM-Exit Instruction Info is considered incomplete for use by a VMM that enables Intel® APX for guest usage, since all regID fields will contain legacy, truncated 4-bit regIDs, instead of full 5-bit regIDs. As such, an Intel® APX-enabled VMM should *only* use and rely on VM-Exit Extended Instruction Info. A VMM that does not enable Intel® APX for guest usage is free to use the legacy VM-Exit Instruction Info, since it is informationally complete if Intel® APX is not enabled.
- Any instruction which has a defined VM Exit Qualification field which contains regID info will continue to populate this info in a legacy-compatible way, although the defined format of this field adds an additional regID bit that had been previously un-defined/reserved. As such, an Intel® APX-enabled VMM should use this field according to the new format, so that it considers a potential 5-bit regID. A non-Intel® APX enabled VMM is free to continue using the legacy definition of the field, since lack of Intel® APX enabling will guarantee that regIDs are only 4-bits, maximum.

Any Intel® APX-aware VMM can use this new EII field to find the full 5-bit regIDs that correspond to decoded reg operands of existing instructions. A non-Intel® APX-enabled VMM (which has not enabled Intel® APX and is therefore not responsible for managing EGPRs) can continue to use the legacy VM Exit Instruction Info field, as it always had previously.

In all VMCS fields, the 5-bit regID encodings of each reg-field are represented in Figure 3.12.

Figure 3.12: VMCS RegID Encodings

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0. RAX | 4. RSP | 8. R8 | 12. R12 | 16. R16 | 20. R20 | 24. R24 | 28. R28 |
| 1. RCX | 5. RBP | 9. R9 | 13. R13 | 17. R17 | 21. R21 | 25. R25 | 29. R29 |
| 2. RDX | 6. RSI | 10. R10 | 14. R14 | 18. R18 | 22. R22 | 26. R26 | 30. R30 |
| 3. RBX | 7. RDI | 11. R11 | 15. R15 | 19. R19 | 23. R23 | 27. R27 | 31. R31 |

The encoding of the new, 64-bit, VM-Exit Extended Instruction Information (EII) VMCS field is 0x2406/0x2407, and the format of this field is shown in Table 3.11

The VM exit qualification field is populated with regID info in several types of instruction exits, namely MOV CR, MOV DR, LMSW, CLTS. This VMCS field will be extended ”in-place” with previously reserved bits containing new meanings in order to indicate the full regID used in these instructions as shown in Figure 3.12 and Figure 3.13.

**3.1.4.4.2 Intel® TDX** Intel® TDX (Intel® Trust Domain Extensions) has similar interactions with Intel® APX as Intel® VMX does.

| Instruction | Use EGPRs | Use NDD | Use NF |
|---|---|---|---|
| CLTS | No | No | No |
| CPUID | No | No | No |
| ENCLS | No | No | No |
| GETSEC | No | No | No |
| HLT | No | No | No |
| IN | No | No | No |
| INS | No | No | No |
| INVEPT | Yes | No | No |
| INVPCID | Yes | No | No |
| INVVPID | Yes | No | No |
| LIDT | Yes | No | No |
| LGDT | Yes | No | No |
| LLDT | Yes | No | No |
| LMSW | Yes | No | No |
| LOADIWKEY | No | No | No |
| LTR | Yes | No | No |
| MONITOR | No | No | No |
| MOV CR | Yes | No | No |
| MOV DR | Yes | No | No |
| MWAIT | No | No | No |
| OUT | No | No | No |
| OUTS | No | No | No |
| PAUSE | No | No | No |
| PCONFIG | No | No | No |
| RDMSR | No | No | No |
| RDPMC | No | No | No |
| RDRAND | Yes | No | No |
| RDSEED | Yes | No | No |
| RDTSC | No | No | No |
| RDTSP | No | No | No |
| RSM | No | No | No |
| SGDT | Yes | No | No |
| SIDT | Yes | No | No |
| SLDT | Yes | No | No |
| STR | Yes | No | No |
| TPAUSE | Yes | No | No |
| UMWAIT | Yes | No | No |
| VMCALL | No | No | No |
| VMCLEAR | Yes | No | No |
| VMLAUNCH | No | No | No |
| VMPTRLD | Yes | No | No |
| VMPTRST | Yes | No | No |
| VMREAD | Yes | No | No |
| VMRESUME | No | No | No |
| VMWRITE | Yes | No | No |
| VMXOFF | No | No | No |
| VMXON | Yes | No | No |
| WBINVD | No | No | No |
| WBNOINVD | No | No | No |
| WRMSR | No | No | No |
| XRSTORS | No | No | No |
| XSETBV | No | No | No |
| XSAVES | No | No | No |

Table 3.10: Intel® APX Interactions with Instructions which Populate VMCS with Instruction Execution Info

Intel® TDX has an XCR0-derived interface called TDCS.XFAM. Bits in XFAM act as an opt-in for state and ISA controls. Therefore, XFAM[APX_F] acts as a control for enabling Intel® APX within Trust Domains (or TDs), and the XFAM settings are established at TD INIT (TDH.TD.INIT).

Trust Domain flows, namely TDH.VP.ENTER and TDEXIT flows, all use XSAVE/XRSTOR to setup, tear-down, and scrub state. These flows will naturally manage Intel® APX state as necessary. In addition, the Intel® TDX Module will perform EGPR context switching on behalf of TDs, and the Intel® TDX debug state save area is extended to include EGPRs.

**3.1.4.4.3 SMM** System Management Mode (SMM) is not affected by Intel® APX.

SMM entry and SMM exit (RSM) flows are not modified in any way. The SMM State Save Area (SSA) will NOT expand to include architectural Intel® APX state. SMM can choose to use Intel® APX state if desired, and can manage it itself (SMM itself is not entered in 64-bit mode by default).

SMM, in default treatment (i.e., non SMM-Transfer Monitor, STM mode) is entered in 32-bit real mode (CR0.PE = CR0.PG = 0, which can be referred to as "big" real mode, with 4GB segments). It is typical for SMM to quickly transition to 64-bit, IA-32e protected-mode (manually), and at that point, SMM code is free to enable/use features as it sees fit (with manual state preservation to protect non-SMM state)

SMM, in STM (SMM-Transfer Monitor) mode, enters in 64-bit, IA-32e protected-mode by default and can choose to enable and use Intel® APX features in the same fashion.

The Intel® Platform Properties Assessment Module (PPAM), also known as Devil's Gate Rock (DGR), is a newer SMM-limiting technology that enforces architectural limitations on the capabilities of SMM code, but does not alter the software-facing rules of the mode in which SMM is entered, only on the capabilities of SMM code (including restrictions on alteration of certain sensitive CRs and MSRs). These technologies are not altered by Intel® APX.

**3.1.4.4.4 TXT (LT and LT-SX) and SMX** Intel® TXT (Intel® Trusted Execution Technology), also known as Intel® LaGrande Technology (also referred to as LT and LT-SX) and SMX (Safer Mode Extensions) are not impacted by Intel® APX. ACMs (Authenticated Code Modules) are not entered in 64-bit, IA-32e protected-mode by default, and as such, cannot directly use Intel® APX features upon entry.

**3.1.4.4.5 Intel® SGX** Intel® Software Guard Extensions (Intel® SGX) has similar interactions with Intel® APX as Intel® VMX and Intel® TDX does.

Intel® SGX's thread context structures (TCS's) must expand to house the amount of state architecturally usable by an enclave. This may include Intel® APX state, based on the XFRM opt-in interface of Intel® SGX. In addition, Intel® SGX's register scrubbing/restoration (AEX) must also expand to cover the EGPR Intel® APX state.

This extension results in the following:

- The State Save Area (SSA) of Intel® SGX is architectural, with a documented size and contents that are accessible to enclave code.
  - SSA includes an XSAVE area, MISC area, and GPRSX area. The new Intel® APX state will be housed within the XSAVE area (architecturally un-compacted), as the state is XSAVE-enabled, and this insulates Intel® SGX-specific structures from Intel® APX-specific modifications (another case where purposeful encapsulation of Intel® APX arch state via XSAVE is useful).
  - Intel® APX becomes an XFRM-based opt-in (Enabled via SECS.ATTRIBUTES.XFRM[APX_F] = 1). The APX_F enable is required to be set at enclave creation time to enable Intel® APX within the SGX enclave. This allows legacy Intel® Secure Enclaves (which don't use Intel® APX) to continue to use the legacy (smaller) SSA definition without modification.
  - INIT, switching, and AEX (scrub + restore) support Intel® APX EGPR state.

**3.1.4.4.6 Debug** Intel Debug features and functionality are not directly affected by Intel® APX, but are extended in terms of state footprint.

Debug features such as Probe Mode, PSMI, and Crash Dump are extended to support Intel® APX state (i.e., collecting/dumping/reading/writing R16-R31)

Table 3.11: VM-Exit Extended Instruction-Information (EII) VMCS Field

| Bits | Name | Meaning |
|---|---|---|
| 1:0 | Scale | Scaling:<br>• 0: No scaling<br>• 1: Scale by 2<br>• 2: Scale by 4<br>• 3: Scale by 8 (64-bit CPUs only)<br>Undefined for instructions with no index register |
| 3:2 | ASIZE | Address size:<br>• 0: 16-bit<br>• 1: 32-bit<br>• 2: 64-bit (64-bit CPUs only)<br>Other values not used/defined |
| 4 | Mem/Reg | Mem/Reg indicator (0=memory, 1=register) |
| 6:5 | OSIZE | Operand size:<br>• 0: 16-bit<br>• 1: 32-bit<br>• 2: 64-bit (64-bit CPUs only)<br>Other values not used/defined |
| 9:7 | Segment | Segment register:<br>• 0: ES<br>• 1: CS<br>• 2: SS<br>• 3: DS<br>• 4: FS<br>• 5: GS<br>Other values not used/defined |
| 10 | IndexInvalid | Index reg invalid indicator (0=valid, 1=invalid) |
| 11 | BaseInvalid | Base reg invalid indicator (0=valid, 1=invalid) |
| 15:12 | RESERVED | Reserved/un-defined (0's) |
| 20:16 | Reg1 | 5-bit regID for Reg1, if applicable |
| 23:21 | RESERVED | Reserved/un-defined (0's) |
| 28:24 | Index | 5-bit regID for Index, if applicable (IndexInvalid=0) |
| 31:29 | RESERVED | Reserved/un-defined (0's) |
| 36:32 | Base | 5-bit regID for Base, if applicable (BaseInvalid=0) |
| 39:37 | RESERVED | Reserved/un-defined (0's) |
| 44:40 | Reg2 | 5-bit regID for Reg2, if applicable |
| 47:45 | RESERVED | Reserved/un-defined (0's) |
| 63:48 | RESERVED | Reserved/un-defined (0's) |

Table 3.12: Exit Qualification for Control Register Accesses (MOV CR, LMSW, CLTS)

| Bits | Name | Meaning |
|---|---|---|
| 3:0 | CR Number | CR Number:<br>• CLTS: Always 0<br>• LMSW: Always 0<br>• MOV CR: CR RegID, where bit 3 is always 0 on CPUs that don't support Intel 64 and CR8 |
| 5:4 | Access Type | Access Type:<br>• 0: MOV to CR<br>• 1: MOV from CR<br>• 2: CLTS<br>• 3: LMSW |
| 6 | LMSW Operand Type | Mem/Reg indicator (0=register, 1=memory). For CLTS and MOV CR, always 0 |
| 7 | RESERVED | Not currently defined |
| 12:8 | GPR | GPR used for MOV CR:<br>• 5-bit regID, before Intel® APX this was a 4-bit regID, see Figure 3.12<br>For CLTS/LMSW, cleared 0 zero |
| 15:13 | RESERVED | Not currently defined |
| 31:16 | Source Data | Source Data:<br>• LMSW: The LMSW source data<br>• CLTS: cleared to 0<br>• MO CR: cleared to 0 |
| 63:32 | RESERVED | Reserved/un-defined (0's) |

Table 3.13: Exit Qualification for Debug Register Accesses (MOV DR)

| Bits | Name | Meaning |
|---|---|---|
| 2:0 | DR Number | DR Number |
| 3 | RESERVED | Not currently defined |
| 4 | DIRECTION | Direction of access (0 = MOV to DR; 1 = MOV from DR) |
| 7:5 | RESERVED | Not currently defined |
| 12:8 | GPR | GPR used for MOV DR:<br>• 5-bit regID, before Intel® APX this was a 4-bit regID, see Figure 3.12 |
| 63:13 | RESERVED | Reserved/un-defined (0's) |

### 3.1.5 List of EVEX-Promoted Intel® APX Instructions

The table below lists all EVEX-promoted Intel® APX instructions. The table columns have the following meanings:

**FROM** The source space-map of the promoted instruction.

**ND** The allowed value(s) of EVEX.ND.

**NF** The allowed value(s) of EVEX.NF.

**PP** The mandatory prefix, which is one of {NP,66,F2,F3} or NP/66, the last of which means that 66 is interpreted as the OSIZE override and the OSIZE can be 16b, 32b or 64b. For instructions promoted from legacy maps {1,2,3}, the PP value may be different from the one in the original instruction. If that is the case, the old PP is shown in parentheses.

**OPC** The main opcode byte in hexadecimal. For instructions promoted from legacy maps {1,2,3}, the main opcode may be different from the one in the original instruction. If that is the case, the old opcode is shown in parentheses.

**REG** The secondary opcode encoded by ModRM.Reg, if it exists.

**MOD** Some instructions require ModRM.Mod to be either 3 or not 3 (!3).

**ICLASS** The instruction name, or "iclass" in XED terminology.

**OPERANDS** The instruction's operands as a comma-separated list, where the destination operand (if it exists) is the first operand and the following naming conventions are used:

- "i*" denotes an immediate, where "*" is the immediate's width in bits. "iz" means that the width depends on OSIZE: if OSIZE is 16b, then the width is 16b; otherwise, the width is 32b.
- "m*" denotes a memory operand, where "*" is the size of the memory access in bits. "mv" means that the size is the same as OSIZE, which can be 16b, 32b or 64b.
- "r*_?" denotes a GPR operand, where "*" is the size of the GPR access in bits. "rv" means that the size is the same as OSIZE, which can be 16b, 32b or 64b. "ry" means that the size is 32b when OSIZE is 16b or 32b, and 64b when OSIZE is 64b. The "?" indicates the register id (B, R or V) used to encode this operand, where "n" denotes the V register id (to avoid confusion with the size indication "v").
- "k_?" and "xmm_?" denote mask and XMM registers, where "?" has the same meaning as in the last item.
- "cl" denotes the register CL (namely, the lowest byte of RCX) and "1" denotes the constant 1. Both "cl" and "1" are implicit operands and do not actually appear in the instruction encoding. They are used only by promoted legacy shift and rotate instructions.
- "dfv" (default flags value) denote the 4-bit value of EVEX.[OF,SF,ZF,CF] that is assigned to the status flags when the source condition code "scc" evaluates to false in CCMPscc and CTESTscc instructions (see Section 3.1.3.2.1).

The instructions are listed in the following order:

- Instructions promoted from the legacy space are listed before those promoted from the VEX space.
- Among the instructions promoted from the legacy space, those from maps 0 and 1 are listed before those from maps 2 and 3.
- The instructions promoted from maps 0 and 1 of the legacy space are listed in lexicographic order on the tuple (iclass, map, opcode, ND).
- The instructions promoted from maps 2 and 3 of the legacy space and from the VEX space are listed in lexicographic order on the tuple (map, iclass, opcode, ND).

| FROM | ND | NF | PP | OPC | REG | MOD | ICLASS | OPERANDS |
|---|---|---|---|---|---|---|---|---|
| Legacy-map0 | 0 | 0 | NP | 10 | | | ADC | m8/r8_b, r8_r |
| Legacy-map0 | 1 | 0 | NP | 10 | | | ADC | r8_n, m8/r8_b, r8_r |
| Legacy-map0 | 0 | 0 | NP/66 | 11 | | | ADC | mv/rv_b, rv_r |
| Legacy-map0 | 1 | 0 | NP/66 | 11 | | | ADC | rv_n, mv/rv_b, rv_r |
| Legacy-map0 | 0 | 0 | NP | 12 | | | ADC | r8_r, m8/r8_b |
| Legacy-map0 | 1 | 0 | NP | 12 | | | ADC | r8_n, r8_r, m8/r8_b |
| Legacy-map0 | 0 | 0 | NP/66 | 13 | | | ADC | rv_r, mv/rv_b |
| Legacy-map0 | 1 | 0 | NP/66 | 13 | | | ADC | rv_n, rv_r, mv/rv_b |
| Legacy-map0 | 0 | 0 | NP | 80 | 2 | | ADC | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0 | NP | 80 | 2 | | ADC | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0 | NP/66 | 81 | 2 | | ADC | mv/rv_b, iz |
| Legacy-map0 | 1 | 0 | NP/66 | 81 | 2 | | ADC | rv_n, mv/rv_b, iz |
| Legacy-map0 | 0 | 0 | NP/66 | 83 | 2 | | ADC | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0 | NP/66 | 83 | 2 | | ADC | rv_n, mv/rv_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP | 00 | | | ADD | m8/r8_b, r8_r |
| Legacy-map0 | 1 | 0/1 | NP | 00 | | | ADD | r8_n, m8/r8_b, r8_r |
| Legacy-map0 | 0 | 0/1 | NP/66 | 01 | | | ADD | mv/rv_b, rv_r |
| Legacy-map0 | 1 | 0/1 | NP/66 | 01 | | | ADD | rv_n, mv/rv_b, rv_r |
| Legacy-map0 | 0 | 0/1 | NP | 02 | | | ADD | r8_r, m8/r8_b |
| Legacy-map0 | 1 | 0/1 | NP | 02 | | | ADD | r8_n, r8_r, m8/r8_b |
| Legacy-map0 | 0 | 0/1 | NP/66 | 03 | | | ADD | rv_r, mv/rv_b |
| Legacy-map0 | 1 | 0/1 | NP/66 | 03 | | | ADD | rv_n, rv_r, mv/rv_b |
| Legacy-map0 | 0 | 0/1 | NP | 80 | 0 | | ADD | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP | 80 | 0 | | ADD | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP/66 | 81 | 0 | | ADD | mv/rv_b, iz |
| Legacy-map0 | 1 | 0/1 | NP/66 | 81 | 0 | | ADD | rv_n, mv/rv_b, iz |
| Legacy-map0 | 0 | 0/1 | NP/66 | 83 | 0 | | ADD | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP/66 | 83 | 0 | | ADD | rv_n, mv/rv_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP | 20 | | | AND | m8/r8_b, r8_r |
| Legacy-map0 | 1 | 0/1 | NP | 20 | | | AND | r8_n, m8/r8_b, r8_r |
| Legacy-map0 | 0 | 0/1 | NP/66 | 21 | | | AND | mv/rv_b, rv_r |
| Legacy-map0 | 1 | 0/1 | NP/66 | 21 | | | AND | rv_n, mv/rv_b, rv_r |
| Legacy-map0 | 0 | 0/1 | NP | 22 | | | AND | r8_r, m8/r8_b |
| Legacy-map0 | 1 | 0/1 | NP | 22 | | | AND | r8_n, r8_r, m8/r8_b |
| Legacy-map0 | 0 | 0/1 | NP/66 | 23 | | | AND | rv_r, mv/rv_b |
| Legacy-map0 | 1 | 0/1 | NP/66 | 23 | | | AND | rv_n, rv_r, mv/rv_b |
| Legacy-map0 | 0 | 0/1 | NP | 80 | 4 | | AND | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP | 80 | 4 | | AND | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP/66 | 81 | 4 | | AND | mv/rv_b, iz |
| Legacy-map0 | 1 | 0/1 | NP/66 | 81 | 4 | | AND | rv_n, mv/rv_b, iz |
| Legacy-map0 | 0 | 0/1 | NP/66 | 83 | 4 | | AND | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP/66 | 83 | 4 | | AND | rv_n, mv/rv_b, i8 |
| Legacy-map0 | 0 | 0 | NP | 38 | | | CCMPscc | m8/r8_b, r8_r, dfv |
| Legacy-map0 | 0 | 0 | NP/66 | 39 | | | CCMPscc | mv/rv_b, rv_r, dfv |
| Legacy-map0 | 0 | 0 | NP | 3A | | | CCMPscc | r8_r, m8/r8_b, dfv |
| Legacy-map0 | 0 | 0 | NP/66 | 3B | | | CCMPscc | rv_r, mv/rv_b, dfv |
| Legacy-map0 | 0 | 0 | NP | 80 | 7 | | CCMPscc | m8/r8_b, i8, dfv |
| Legacy-map0 | 0 | 0 | NP/66 | 81 | 7 | | CCMPscc | mv/rv_b, iz, dfv |
| Legacy-map0 | 0 | 0 | NP/66 | 83 | 7 | | CCMPscc | mv/rv_b, i8, dfv |
| Legacy-map1 | 0 | 0 | NP/66 | 42 | | | CFCMOVB | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 42 | | | CFCMOVB | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 42 | | | CFCMOVB | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 46 | | | CFCMOVBE | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 46 | | | CFCMOVBE | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 46 | | | CFCMOVBE | rv_n, rv_r, mv/rv_b |

| FROM | ND | NF | PP | OPC | REG | MOD | ICLASS | OPERANDS |
|---|---|---|---|---|---|---|---|---|
| Legacy-map1 | 0 | 0 | NP/66 | 4C | | | CFCMOVL | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 4C | | | CFCMOVL | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 4C | | | CFCMOVL | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 4E | | | CFCMOVLE | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 4E | | | CFCMOVLE | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 4E | | | CFCMOVLE | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 43 | | | CFCMOVNB | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 43 | | | CFCMOVNB | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 43 | | | CFCMOVNB | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 47 | | | CFCMOVNBE | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 47 | | | CFCMOVNBE | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 47 | | | CFCMOVNBE | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 4D | | | CFCMOVNL | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 4D | | | CFCMOVNL | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 4D | | | CFCMOVNL | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 4F | | | CFCMOVNLE | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 4F | | | CFCMOVNLE | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 4F | | | CFCMOVNLE | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 41 | | | CFCMOVNO | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 41 | | | CFCMOVNO | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 41 | | | CFCMOVNO | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 4B | | | CFCMOVNP | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 4B | | | CFCMOVNP | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 4B | | | CFCMOVNP | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 49 | | | CFCMOVNS | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 49 | | | CFCMOVNS | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 49 | | | CFCMOVNS | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 45 | | | CFCMOVNZ | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 45 | | | CFCMOVNZ | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 45 | | | CFCMOVNZ | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 40 | | | CFCMOVO | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 40 | | | CFCMOVO | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 40 | | | CFCMOVO | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 4A | | | CFCMOVP | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 4A | | | CFCMOVP | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 4A | | | CFCMOVP | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 48 | | | CFCMOVS | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 48 | | | CFCMOVS | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 48 | | | CFCMOVS | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 0 | 0 | NP/66 | 44 | | | CFCMOVZ | rv_r, mv/rv_b |
| Legacy-map1 | 0 | 1 | NP/66 | 44 | | | CFCMOVZ | mv/rv_b, rv_r |
| Legacy-map1 | 1 | 1 | NP/66 | 44 | | | CFCMOVZ | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 42 | | | CMOVB | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 46 | | | CMOVBE | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 4C | | | CMOVL | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 4E | | | CMOVLE | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 43 | | | CMOVNB | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 47 | | | CMOVNBE | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 4D | | | CMOVNL | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 4F | | | CMOVNLE | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 41 | | | CMOVNO | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 4B | | | CMOVNP | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 49 | | | CMOVNS | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 45 | | | CMOVNZ | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 40 | | | CMOVO | rv_n, rv_r, mv/rv_b |

| FROM | ND | NF | PP | OPC | REG | MOD | ICLASS | OPERANDS |
|---|---|---|---|---|---|---|---|---|
| Legacy-map1 | 1 | 0 | NP/66 | 4A | | | CMOVP | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 48 | | | CMOVS | rv_n, rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0 | NP/66 | 44 | | | CMOVZ | rv_n, rv_r, mv/rv_b |
| Legacy-map0 | 0 | 0 | NP | 84 | | | CTESTscc | m8/r8_b, r8_r, dfv |
| Legacy-map0 | 0 | 0 | NP/66 | 85 | | | CTESTscc | mv/rv_b, rv_r, dfv |
| Legacy-map0 | 0 | 0 | NP | F6 | 0 | | CTESTscc | m8/r8_b, i8, dfv |
| Legacy-map0 | 0 | 0 | NP | F6 | 1 | | CTESTscc | m8/r8_b, i8, dfv |
| Legacy-map0 | 0 | 0 | NP/66 | F7 | 0 | | CTESTscc | mv/rv_b, iz, dfv |
| Legacy-map0 | 0 | 0 | NP/66 | F7 | 1 | | CTESTscc | mv/rv_b, iz, dfv |
| Legacy-map0 | 0 | 0/1 | NP | FE | 1 | | DEC | m8/r8_b |
| Legacy-map0 | 1 | 0/1 | NP | FE | 1 | | DEC | r8_n, m8/r8_b |
| Legacy-map0 | 0 | 0/1 | NP/66 | FF | 1 | | DEC | mv/rv_b |
| Legacy-map0 | 1 | 0/1 | NP/66 | FF | 1 | | DEC | rv_n, mv/rv_b |
| Legacy-map0 | 0 | 0/1 | NP | F6 | 6 | | DIV | m8/r8_b |
| Legacy-map0 | 0 | 0/1 | NP/66 | F7 | 6 | | DIV | mv/rv_b |
| Legacy-map0 | 0 | 0/1 | NP | F6 | 7 | | IDIV | m8/r8_b |
| Legacy-map0 | 0 | 0/1 | NP/66 | F7 | 7 | | IDIV | mv/rv_b |
| Legacy-map0 | 0/1 | 0/1 | NP/66 | 69 | | | IMUL | rv_r, mv/rv_b, iz |
| Legacy-map0 | 0/1 | 0/1 | NP/66 | 6B | | | IMUL | rv_r, mv/rv_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP | F6 | 5 | | IMUL | m8/r8_b |
| Legacy-map0 | 0 | 0/1 | NP/66 | F7 | 5 | | IMUL | mv/rv_b |
| Legacy-map1 | 0 | 0/1 | NP/66 | AF | | | IMUL | rv_r, mv/rv_b |
| Legacy-map1 | 1 | 0/1 | NP/66 | AF | | | IMUL | rv_n, rv_r, mv/rv_b |
| Legacy-map0 | 0 | 0/1 | NP | FE | 0 | | INC | m8/r8_b |
| Legacy-map0 | 1 | 0/1 | NP | FE | 0 | | INC | r8_n, m8/r8_b |
| Legacy-map0 | 0 | 0/1 | NP/66 | FF | 0 | | INC | mv/rv_b |
| Legacy-map0 | 1 | 0/1 | NP/66 | FF | 0 | | INC | rv_n, mv/rv_b |
| Legacy-map1 | 0 | 0/1 | NP/66 (F3) | F5 (BD) | | | LZCNT | rv_r, mv/rv_b |
| Legacy-map0 | 0 | 0/1 | NP | F6 | 4 | | MUL | m8/r8_b |
| Legacy-map0 | 0 | 0/1 | NP/66 | F7 | 4 | | MUL | mv/rv_b |
| Legacy-map0 | 0 | 0/1 | NP | F6 | 3 | | NEG | m8/r8_b |
| Legacy-map0 | 1 | 0/1 | NP | F6 | 3 | | NEG | r8_n, m8/r8_b |
| Legacy-map0 | 0 | 0/1 | NP/66 | F7 | 3 | | NEG | mv/rv_b |
| Legacy-map0 | 1 | 0/1 | NP/66 | F7 | 3 | | NEG | rv_n, mv/rv_b |
| Legacy-map0 | 0 | 0 | NP | F6 | 2 | | NOT | m8/r8_b |
| Legacy-map0 | 1 | 0 | NP | F6 | 2 | | NOT | r8_n, m8/r8_b |
| Legacy-map0 | 0 | 0 | NP/66 | F7 | 2 | | NOT | mv/rv_b |
| Legacy-map0 | 1 | 0 | NP/66 | F7 | 2 | | NOT | rv_n, mv/rv_b |
| Legacy-map0 | 0 | 0/1 | NP | 08 | | | OR | m8/r8_b, r8_r |
| Legacy-map0 | 1 | 0/1 | NP | 08 | | | OR | r8_n, m8/r8_b, r8_r |
| Legacy-map0 | 0 | 0/1 | NP/66 | 09 | | | OR | mv/rv_b, rv_r |
| Legacy-map0 | 1 | 0/1 | NP/66 | 09 | | | OR | rv_n, mv/rv_b, rv_r |
| Legacy-map0 | 0 | 0/1 | NP | 0A | | | OR | r8_r, m8/r8_b |
| Legacy-map0 | 1 | 0/1 | NP | 0A | | | OR | r8_n, r8_r, m8/r8_b |
| Legacy-map0 | 0 | 0/1 | NP/66 | 0B | | | OR | rv_r, mv/rv_b |
| Legacy-map0 | 1 | 0/1 | NP/66 | 0B | | | OR | rv_n, rv_r, mv/rv_b |
| Legacy-map0 | 0 | 0/1 | NP | 80 | 1 | | OR | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP | 80 | 1 | | OR | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP/66 | 81 | 1 | | OR | mv/rv_b, iz |
| Legacy-map0 | 1 | 0/1 | NP/66 | 81 | 1 | | OR | rv_n, mv/rv_b, iz |
| Legacy-map0 | 0 | 0/1 | NP/66 | 83 | 1 | | OR | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP/66 | 83 | 1 | | OR | rv_n, mv/rv_b, i8 |
| Legacy-map0 | 1 | 0 | NP | 8F | 0 | 3 | POP2 | r64_n, r64_b |
| Legacy-map1 | 0 | 0/1 | NP/66 (F3) | 88 (B8) | | | POPCNT | rv_r, mv/rv_b |
| Legacy-map0 | 1 | 0 | NP | FF | 6 | 3 | PUSH2 | r64_n, r64_b |

| FROM | ND | NF | PP | OPC | REG | MOD | ICLASS | OPERANDS |
|---|---|---|---|---|---|---|---|---|
| Legacy-map0 | 0 | 0 | NP | C0 | 2 | | RCL | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0 | NP | C0 | 2 | | RCL | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0 | NP/66 | C1 | 2 | | RCL | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0 | NP/66 | C1 | 2 | | RCL | rv_n, mv/rv_b, i8 |
| Legacy-map0 | 0 | 0 | NP | D0 | 2 | | RCL | m8/r8_b, 1 |
| Legacy-map0 | 1 | 0 | NP | D0 | 2 | | RCL | r8_n, m8/r8_b, 1 |
| Legacy-map0 | 0 | 0 | NP/66 | D1 | 2 | | RCL | mv/rv_b, 1 |
| Legacy-map0 | 1 | 0 | NP/66 | D1 | 2 | | RCL | rv_n, mv/rv_b, 1 |
| Legacy-map0 | 0 | 0 | NP | D2 | 2 | | RCL | m8/r8_b, cl |
| Legacy-map0 | 1 | 0 | NP | D2 | 2 | | RCL | r8_n, m8/r8_b, cl |
| Legacy-map0 | 0 | 0 | NP/66 | D3 | 2 | | RCL | mv/rv_b, cl |
| Legacy-map0 | 1 | 0 | NP/66 | D3 | 2 | | RCL | rv_n, mv/rv_b, cl |
| Legacy-map0 | 0 | 0 | NP | C0 | 3 | | RCR | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0 | NP | C0 | 3 | | RCR | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0 | NP/66 | C1 | 3 | | RCR | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0 | NP/66 | C1 | 3 | | RCR | rv_n, mv/rv_b, i8 |
| Legacy-map0 | 0 | 0 | NP | D0 | 3 | | RCR | m8/r8_b, 1 |
| Legacy-map0 | 1 | 0 | NP | D0 | 3 | | RCR | r8_n, m8/r8_b, 1 |
| Legacy-map0 | 0 | 0 | NP/66 | D1 | 3 | | RCR | mv/rv_b, 1 |
| Legacy-map0 | 1 | 0 | NP/66 | D1 | 3 | | RCR | rv_n, mv/rv_b, 1 |
| Legacy-map0 | 0 | 0 | NP | D2 | 3 | | RCR | m8/r8_b, cl |
| Legacy-map0 | 1 | 0 | NP | D2 | 3 | | RCR | r8_n, m8/r8_b, cl |
| Legacy-map0 | 0 | 0 | NP/66 | D3 | 3 | | RCR | mv/rv_b, cl |
| Legacy-map0 | 1 | 0 | NP/66 | D3 | 3 | | RCR | rv_n, mv/rv_b, cl |
| Legacy-map0 | 0 | 0/1 | NP | C0 | 0 | | ROL | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP | C0 | 0 | | ROL | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP/66 | C1 | 0 | | ROL | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP/66 | C1 | 0 | | ROL | rv_n, mv/rv_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP | D0 | 0 | | ROL | m8/r8_b, 1 |
| Legacy-map0 | 1 | 0/1 | NP | D0 | 0 | | ROL | r8_n, m8/r8_b, 1 |
| Legacy-map0 | 0 | 0/1 | NP/66 | D1 | 0 | | ROL | mv/rv_b, 1 |
| Legacy-map0 | 1 | 0/1 | NP/66 | D1 | 0 | | ROL | rv_n, mv/rv_b, 1 |
| Legacy-map0 | 0 | 0/1 | NP | D2 | 0 | | ROL | m8/r8_b, cl |
| Legacy-map0 | 1 | 0/1 | NP | D2 | 0 | | ROL | r8_n, m8/r8_b, cl |
| Legacy-map0 | 0 | 0/1 | NP/66 | D3 | 0 | | ROL | mv/rv_b, cl |
| Legacy-map0 | 1 | 0/1 | NP/66 | D3 | 0 | | ROL | rv_n, mv/rv_b, cl |
| Legacy-map0 | 0 | 0/1 | NP | C0 | 1 | | ROR | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP | C0 | 1 | | ROR | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP/66 | C1 | 1 | | ROR | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP/66 | C1 | 1 | | ROR | rv_n, mv/rv_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP | D0 | 1 | | ROR | m8/r8_b, 1 |
| Legacy-map0 | 1 | 0/1 | NP | D0 | 1 | | ROR | r8_n, m8/r8_b, 1 |
| Legacy-map0 | 0 | 0/1 | NP/66 | D1 | 1 | | ROR | mv/rv_b, 1 |
| Legacy-map0 | 1 | 0/1 | NP/66 | D1 | 1 | | ROR | rv_n, mv/rv_b, 1 |
| Legacy-map0 | 0 | 0/1 | NP | D2 | 1 | | ROR | m8/r8_b, cl |
| Legacy-map0 | 1 | 0/1 | NP | D2 | 1 | | ROR | r8_n, m8/r8_b, cl |
| Legacy-map0 | 0 | 0/1 | NP/66 | D3 | 1 | | ROR | mv/rv_b, cl |
| Legacy-map0 | 1 | 0/1 | NP/66 | D3 | 1 | | ROR | rv_n, mv/rv_b, cl |
| Legacy-map0 | 0 | 0/1 | NP | C0 | 7 | | SAR | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP | C0 | 7 | | SAR | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP/66 | C1 | 7 | | SAR | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP/66 | C1 | 7 | | SAR | rv_n, mv/rv_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP | D0 | 7 | | SAR | m8/r8_b, 1 |
| Legacy-map0 | 1 | 0/1 | NP | D0 | 7 | | SAR | r8_n, m8/r8_b, 1 |
| Legacy-map0 | 0 | 0/1 | NP/66 | D1 | 7 | | SAR | mv/rv_b, 1 |

| FROM | ND | NF | PP | OPC | REG | MOD | ICLASS | OPERANDS |
|---|---|---|---|---|---|---|---|---|
| Legacy-map0 | 1 | 0/1 | NP/66 | D1 | 7 | | SAR | rv_n, mv/rv_b, 1 |
| Legacy-map0 | 0 | 0/1 | NP | D2 | 7 | | SAR | m8/r8_b, cl |
| Legacy-map0 | 1 | 0/1 | NP | D2 | 7 | | SAR | r8_n, m8/r8_b, cl |
| Legacy-map0 | 0 | 0/1 | NP/66 | D3 | 7 | | SAR | mv/rv_b, cl |
| Legacy-map0 | 1 | 0/1 | NP/66 | D3 | 7 | | SAR | rv_n, mv/rv_b, cl |
| Legacy-map0 | 0 | 0 | NP | 18 | | | SBB | m8/r8_b, r8_r |
| Legacy-map0 | 1 | 0 | NP | 18 | | | SBB | r8_n, m8/r8_b, r8_r |
| Legacy-map0 | 0 | 0 | NP/66 | 19 | | | SBB | mv/rv_b, rv_r |
| Legacy-map0 | 1 | 0 | NP/66 | 19 | | | SBB | rv_n, mv/rv_b, rv_r |
| Legacy-map0 | 0 | 0 | NP | 1A | | | SBB | r8_r, m8/r8_b |
| Legacy-map0 | 1 | 0 | NP | 1A | | | SBB | r8_n, r8_r, m8/r8_b |
| Legacy-map0 | 0 | 0 | NP/66 | 1B | | | SBB | rv_r, mv/rv_b |
| Legacy-map0 | 1 | 0 | NP/66 | 1B | | | SBB | rv_n, rv_r, mv/rv_b |
| Legacy-map0 | 0 | 0 | NP | 80 | 3 | | SBB | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0 | NP | 80 | 3 | | SBB | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0 | NP/66 | 81 | 3 | | SBB | mv/rv_b, iz |
| Legacy-map0 | 1 | 0 | NP/66 | 81 | 3 | | SBB | rv_n, mv/rv_b, iz |
| Legacy-map0 | 0 | 0 | NP/66 | 83 | 3 | | SBB | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0 | NP/66 | 83 | 3 | | SBB | rv_n, mv/rv_b, i8 |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 42 (92) | | | SETB | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 46 (96) | | | SETBE | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 4C (9C) | | | SETL | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 4E (9E) | | | SETLE | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 43 (93) | | | SETNB | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 47 (97) | | | SETNBE | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 4D (9D) | | | SETNL | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 4F (9F) | | | SETNLE | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 41 (91) | | | SETNO | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 4B (9B) | | | SETNP | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 49 (99) | | | SETNS | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 45 (95) | | | SETNZ | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 40 (90) | | | SETO | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 4A (9A) | | | SETP | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 48 (98) | | | SETS | m8/r8_b |
| Legacy-map1 | 0/1 | 0 | F2 (NP) | 44 (94) | | | SETZ | m8/r8_b |
| Legacy-map0 | 0 | 0/1 | NP | C0 | 4 | | SHL | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP | C0 | 4 | | SHL | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP | C0 | 6 | | SHL | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP | C0 | 6 | | SHL | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP/66 | C1 | 4 | | SHL | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP/66 | C1 | 4 | | SHL | rv_n, mv/rv_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP/66 | C1 | 6 | | SHL | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP/66 | C1 | 6 | | SHL | rv_n, mv/rv_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP | D0 | 4 | | SHL | m8/r8_b, 1 |
| Legacy-map0 | 1 | 0/1 | NP | D0 | 4 | | SHL | r8_n, m8/r8_b, 1 |
| Legacy-map0 | 0 | 0/1 | NP | D0 | 6 | | SHL | m8/r8_b, 1 |
| Legacy-map0 | 1 | 0/1 | NP | D0 | 6 | | SHL | r8_n, m8/r8_b, 1 |
| Legacy-map0 | 0 | 0/1 | NP/66 | D1 | 4 | | SHL | mv/rv_b, 1 |
| Legacy-map0 | 1 | 0/1 | NP/66 | D1 | 4 | | SHL | rv_n, mv/rv_b, 1 |
| Legacy-map0 | 0 | 0/1 | NP/66 | D1 | 6 | | SHL | mv/rv_b, 1 |
| Legacy-map0 | 1 | 0/1 | NP/66 | D1 | 6 | | SHL | rv_n, mv/rv_b, 1 |
| Legacy-map0 | 0 | 0/1 | NP | D2 | 4 | | SHL | m8/r8_b, cl |
| Legacy-map0 | 1 | 0/1 | NP | D2 | 4 | | SHL | r8_n, m8/r8_b, cl |
| Legacy-map0 | 0 | 0/1 | NP | D2 | 6 | | SHL | m8/r8_b, cl |
| Legacy-map0 | 1 | 0/1 | NP | D2 | 6 | | SHL | r8_n, m8/r8_b, cl |

| FROM | ND | NF | PP | OPC | REG | MOD | ICLASS | OPERANDS |
|---|---|---|---|---|---|---|---|---|
| Legacy-map0 | 0 | 0/1 | NP/66 | D3 | 4 | | SHL | mv/rv_b, cl |
| Legacy-map0 | 1 | 0/1 | NP/66 | D3 | 4 | | SHL | rv_n, mv/rv_b, cl |
| Legacy-map0 | 0 | 0/1 | NP/66 | D3 | 6 | | SHL | mv/rv_b, cl |
| Legacy-map0 | 1 | 0/1 | NP/66 | D3 | 6 | | SHL | rv_n, mv/rv_b, cl |
| Legacy-map1 | 0 | 0/1 | NP/66 | 24 (A4) | | | SHLD | mv/rv_b, rv_r, i8 |
| Legacy-map1 | 1 | 0/1 | NP/66 | 24 (A4) | | | SHLD | rv_n, mv/rv_b, rv_r, i8 |
| Legacy-map1 | 0 | 0/1 | NP/66 | A5 | | | SHLD | mv/rv_b, rv_r, cl |
| Legacy-map1 | 1 | 0/1 | NP/66 | A5 | | | SHLD | rv_n, mv/rv_b, rv_r, cl |
| Legacy-map0 | 0 | 0/1 | NP | C0 | 5 | | SHR | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP | C0 | 5 | | SHR | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP/66 | C1 | 5 | | SHR | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP/66 | C1 | 5 | | SHR | rv_n, mv/rv_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP | D0 | 5 | | SHR | m8/r8_b, 1 |
| Legacy-map0 | 1 | 0/1 | NP | D0 | 5 | | SHR | r8_n, m8/r8_b, 1 |
| Legacy-map0 | 0 | 0/1 | NP/66 | D1 | 5 | | SHR | mv/rv_b, 1 |
| Legacy-map0 | 1 | 0/1 | NP/66 | D1 | 5 | | SHR | rv_n, mv/rv_b, 1 |
| Legacy-map0 | 0 | 0/1 | NP | D2 | 5 | | SHR | m8/r8_b, cl |
| Legacy-map0 | 1 | 0/1 | NP | D2 | 5 | | SHR | r8_n, m8/r8_b, cl |
| Legacy-map0 | 0 | 0/1 | NP/66 | D3 | 5 | | SHR | mv/rv_b, cl |
| Legacy-map0 | 1 | 0/1 | NP/66 | D3 | 5 | | SHR | rv_n, mv/rv_b, cl |
| Legacy-map1 | 0 | 0/1 | NP/66 | 2C (AC) | | | SHRD | mv/rv_b, rv_r, i8 |
| Legacy-map1 | 1 | 0/1 | NP/66 | 2C (AC) | | | SHRD | rv_n, mv/rv_b, rv_r, i8 |
| Legacy-map1 | 0 | 0/1 | NP/66 | AD | | | SHRD | mv/rv_b, rv_r, cl |
| Legacy-map1 | 1 | 0/1 | NP/66 | AD | | | SHRD | rv_n, mv/rv_b, rv_r, cl |
| Legacy-map0 | 0 | 0/1 | NP | 28 | | | SUB | m8/r8_b, r8_r |
| Legacy-map0 | 1 | 0/1 | NP | 28 | | | SUB | r8_n, m8/r8_b, r8_r |
| Legacy-map0 | 0 | 0/1 | NP/66 | 29 | | | SUB | mv/rv_b, rv_r |
| Legacy-map0 | 1 | 0/1 | NP/66 | 29 | | | SUB | rv_n, mv/rv_b, rv_r |
| Legacy-map0 | 0 | 0/1 | NP | 2A | | | SUB | r8_r, m8/r8_b |
| Legacy-map0 | 1 | 0/1 | NP | 2A | | | SUB | r8_n, r8_r, m8/r8_b |
| Legacy-map0 | 0 | 0/1 | NP/66 | 2B | | | SUB | rv_r, mv/rv_b |
| Legacy-map0 | 1 | 0/1 | NP/66 | 2B | | | SUB | rv_n, rv_r, mv/rv_b |
| Legacy-map0 | 0 | 0/1 | NP | 80 | 5 | | SUB | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP | 80 | 5 | | SUB | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP/66 | 81 | 5 | | SUB | mv/rv_b, iz |
| Legacy-map0 | 1 | 0/1 | NP/66 | 81 | 5 | | SUB | rv_n, mv/rv_b, iz |
| Legacy-map0 | 0 | 0/1 | NP/66 | 83 | 5 | | SUB | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP/66 | 83 | 5 | | SUB | rv_n, mv/rv_b, i8 |
| Legacy-map1 | 0 | 0/1 | NP/66 (F3) | F4 (BC) | | | TZCNT | rv_r, mv/rv_b |
| Legacy-map0 | 0 | 0/1 | NP | 30 | | | XOR | m8/r8_b, r8_r |
| Legacy-map0 | 1 | 0/1 | NP | 30 | | | XOR | r8_n, m8/r8_b, r8_r |
| Legacy-map0 | 0 | 0/1 | NP/66 | 31 | | | XOR | mv/rv_b, rv_r |
| Legacy-map0 | 1 | 0/1 | NP/66 | 31 | | | XOR | rv_n, mv/rv_b, rv_r |
| Legacy-map0 | 0 | 0/1 | NP | 32 | | | XOR | r8_r, m8/r8_b |
| Legacy-map0 | 1 | 0/1 | NP | 32 | | | XOR | r8_n, r8_r, m8/r8_b |
| Legacy-map0 | 0 | 0/1 | NP/66 | 33 | | | XOR | rv_r, mv/rv_b |
| Legacy-map0 | 1 | 0/1 | NP/66 | 33 | | | XOR | rv_n, rv_r, mv/rv_b |
| Legacy-map0 | 0 | 0/1 | NP | 80 | 6 | | XOR | m8/r8_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP | 80 | 6 | | XOR | r8_n, m8/r8_b, i8 |
| Legacy-map0 | 0 | 0/1 | NP/66 | 81 | 6 | | XOR | mv/rv_b, iz |
| Legacy-map0 | 1 | 0/1 | NP/66 | 81 | 6 | | XOR | rv_n, mv/rv_b, iz |
| Legacy-map0 | 0 | 0/1 | NP/66 | 83 | 6 | | XOR | mv/rv_b, i8 |
| Legacy-map0 | 1 | 0/1 | NP/66 | 83 | 6 | | XOR | rv_n, mv/rv_b, i8 |
| Legacy-map2 | 0 | 0 | 66 | 66 (F6) | | | ADCX | r32_r/r64_r, m32/m64/r32_b/r64_b |
| Legacy-map2 | 1 | 0 | 66 | 66 (F6) | | | ADCX | r32_n/r64_n, r32_r/r64_r, m32/m64/r32_b/r64_b |

| FROM | ND | NF | PP | OPC | REG | MOD | ICLASS | OPERANDS |
|---|---|---|---|---|---|---|---|---|
| Legacy-map2 | 0 | 0 | F3 | 66 (F6) | | | ADOX | r32_r/r64_r, m32/m64/r32_b/r64_b |
| Legacy-map2 | 1 | 0 | F3 | 66 (F6) | | | ADOX | r32_n/r64_n, r32_r/r64_r, m32/m64/r32_b/r64_b |
| Legacy-map2 | 0 | 0 | F3 | DD | | !3 | AESDEC128KL | xmm_r, m384 |
| Legacy-map2 | 0 | 0 | F3 | DF | | !3 | AESDEC256KL | xmm_r, m512 |
| Legacy-map2 | 0 | 0 | F3 | D8 | 1 | !3 | AESDECWIDE128KL | m384 |
| Legacy-map2 | 0 | 0 | F3 | D8 | 3 | !3 | AESDECWIDE256KL | m512 |
| Legacy-map2 | 0 | 0 | F3 | DC | | !3 | AESENC128KL | xmm_r, m384 |
| Legacy-map2 | 0 | 0 | F3 | DE | | !3 | AESENC256KL | xmm_r, m512 |
| Legacy-map2 | 0 | 0 | F3 | D8 | 0 | !3 | AESENCWIDE128KL | m384 |
| Legacy-map2 | 0 | 0 | F3 | D8 | 2 | !3 | AESENCWIDE256KL | m512 |
| Legacy-map2 | 0 | 0 | NP (F2) | F0 | | | CRC32 | ry_r, m8/r8_b |
| Legacy-map2 | 0 | 0 | NP/66 (F2) | F1 | | | CRC32 | ry_r, mv/rv_b |
| Legacy-map2 | 0 | 0 | F3 | DA (FA) | | 3 | ENCODEKEY128 | r32_r, r32_b |
| Legacy-map2 | 0 | 0 | F3 | DB (FB) | | 3 | ENCODEKEY256 | r32_r, r32_b |
| Legacy-map2 | 0 | 0 | F2 | F8 | | !3 | ENQCMD | r64_r, m512 |
| Legacy-map2 | 0 | 0 | F3 | F8 | | !3 | ENQCMDS | r64_r, m512 |
| Legacy-map2 | 0 | 0 | F3 (66) | F0 (80) | | !3 | INVEPT | r64_r, m128 |
| Legacy-map2 | 0 | 0 | F3 (66) | F2 (82) | | !3 | INVPCID | r64_r, m128 |
| Legacy-map2 | 0 | 0 | F3 (66) | F1 (81) | | !3 | INVVPID | r64_r, m128 |
| Legacy-map2 | 0 | 0 | NP/66 | 60 (F0) | | | MOVBE | rv_r, mv/rv_b |
| Legacy-map2 | 0 | 0 | NP/66 | 61 (F1) | | | MOVBE | mv/rv_b, rv_r |
| Legacy-map2 | 0 | 0 | 66 | F8 | | !3 | MOVDIR64B | r64_r, m512 |
| Legacy-map2 | 0 | 0 | NP | F9 | | !3 | MOVDIRI | m32/m64, r32_r/r64_r |
| Legacy-map2 | 0 | 0 | NP | D9 (C9) | | | SHA1MSG1 | xmm_r, m128/xmm_b |
| Legacy-map2 | 0 | 0 | NP | DA (CA) | | | SHA1MSG2 | xmm_r, m128/xmm_b |
| Legacy-map2 | 0 | 0 | NP | D8 (C8) | | | SHA1NEXTE | xmm_r, m128/xmm_b |
| Legacy-map2 | 0 | 0 | NP | DC (CC) | | | SHA256MSG1 | xmm_r, m128/xmm_b |
| Legacy-map2 | 0 | 0 | NP | DD (CD) | | | SHA256MSG2 | xmm_r, m128/xmm_b |
| Legacy-map2 | 0 | 0 | NP | DB (CB) | | | SHA256RNDS2 | xmm_r, m128/xmm_b |
| Legacy-map2 | 0 | 0 | NP | 66 (F6) | | !3 | WRSSD | m32, r32_r |
| Legacy-map2 | 0 | 0 | NP | 66 (F6) | | !3 | WRSSQ | m64, r64_r |
| Legacy-map2 | 0 | 0 | 66 | 65 (F5) | | !3 | WRUSSD | m32, r32_r |
| Legacy-map2 | 0 | 0 | 66 | 65 (F5) | | !3 | WRUSSQ | m64, r64_r |
| Legacy-map3 | 0 | 0 | NP | D4 (CC) | | | SHA1RNDS4 | xmm_r, m128/xmm_b, i8 |
| VEX-map1 | 0 | 0 | 66 | 90 | | | KMOVB | k_r, k_b/m8 |
| VEX-map1 | 0 | 0 | 66 | 91 | | !3 | KMOVB | m8, k_r |
| VEX-map1 | 0 | 0 | 66 | 92 | | 3 | KMOVB | k_r, r32_b |
| VEX-map1 | 0 | 0 | 66 | 93 | | 3 | KMOVB | r32_r, k_b |
| VEX-map1 | 0 | 0 | 66 | 90 | | | KMOVD | k_r, k_b/m32 |
| VEX-map1 | 0 | 0 | 66 | 91 | | !3 | KMOVD | m32, k_r |
| VEX-map1 | 0 | 0 | F2 | 92 | | 3 | KMOVD | k_r, r32_b |
| VEX-map1 | 0 | 0 | F2 | 93 | | 3 | KMOVD | r32_r, k_b |
| VEX-map1 | 0 | 0 | NP | 90 | | | KMOVQ | k_r, k_b/m64 |
| VEX-map1 | 0 | 0 | NP | 91 | | !3 | KMOVQ | m64, k_r |
| VEX-map1 | 0 | 0 | F2 | 92 | | 3 | KMOVQ | k_r, r64_b |
| VEX-map1 | 0 | 0 | F2 | 93 | | 3 | KMOVQ | r64_r, k_b |
| VEX-map1 | 0 | 0 | NP | 90 | | | KMOVW | k_r, k_b/m16 |
| VEX-map1 | 0 | 0 | NP | 91 | | !3 | KMOVW | m16, k_r |
| VEX-map1 | 0 | 0 | NP | 92 | | 3 | KMOVW | k_r, r32_b |
| VEX-map1 | 0 | 0 | NP | 93 | | 3 | KMOVW | r32_r, k_b |
| VEX-map2 | 0 | 0/1 | NP | F2 | | | ANDN | r32_r/r64_r, r32_n/r64_n, m32/m64/r32_b/r64_b |
| VEX-map2 | 0 | 0/1 | NP | F7 | | | BEXTR | r32_r/r64_r, m32/m64/r32_b/r64_b, r32_n/r64_n |
| VEX-map2 | 0 | 0/1 | NP | F3 | 3 | | BLSI | r32_n/r64_n, m32/m64/r32_b/r64_b |
| VEX-map2 | 0 | 0/1 | NP | F3 | 2 | | BLSMSK | r32_n/r64_n, m32/m64/r32_b/r64_b |
| VEX-map2 | 0 | 0/1 | NP | F3 | 1 | | BLSR | r32_n/r64_n, m32/m64/r32_b/r64_b |

| FROM | ND | NF | PP | OPC | REG | MOD | ICLASS | OPERANDS |
|---|---|---|---|---|---|---|---|---|
| VEX-map2 | 0 | 0/1 | NP | F5 | | | BZHI | r32_r/r64_r, m32/m64/r32_b/r64_b, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | E6 | | !3 | CMPBEXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | E2 | | !3 | CMPBXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | EE | | !3 | CMPLEXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | EC | | !3 | CMPLXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | E7 | | !3 | CMPNBEXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | E3 | | !3 | CMPNBXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | EF | | !3 | CMPNLEXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | ED | | !3 | CMPNLXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | E1 | | !3 | CMPNOXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | EB | | !3 | CMPNPXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | E9 | | !3 | CMPNSXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | E5 | | !3 | CMPNZXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | E0 | | !3 | CMPOXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | EA | | !3 | CMPPXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | E8 | | !3 | CMPSXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | E4 | | !3 | CMPZXADD | m32/m64, r32_r/r64_r, r32_n/r64_n |
| VEX-map2 | 0 | 0 | NP | 49 | 0 | !3 | LDTILECFG | m512 |
| VEX-map2 | 0 | 0 | F2 | F6 | | | MULX | r32_r/r64_r, r32_n/r64_n, m32/m64/r32_b/r64_b |
| VEX-map2 | 0 | 0 | F2 | F5 | | | PDEP | r32_r/r64_r, r32_n/r64_n, m32/m64/r32_b/r64_b |
| VEX-map2 | 0 | 0 | F3 | F5 | | | PEXT | r32_r/r64_r, r32_n/r64_n, m32/m64/r32_b/r64_b |
| VEX-map2 | 0 | 0 | F3 | F7 | | | SARX | r32_r/r64_r, m32/m64/r32_b/r64_b, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | F7 | | | SHLX | r32_r/r64_r, m32/m64/r32_b/r64_b, r32_n/r64_n |
| VEX-map2 | 0 | 0 | F2 | F7 | | | SHRX | r32_r/r64_r, m32/m64/r32_b/r64_b, r32_n/r64_n |
| VEX-map2 | 0 | 0 | 66 | 49 | 0 | !3 | STTILECFG | m512 |
| VEX-map2 | 0 | 0 | F2 | 4B | | !3 | TILELOADD | tmm1, sibmem |
| VEX-map2 | 0 | 0 | 66 | 4B | | !3 | TILELOADDDT1 | tmm1, sibmem |
| VEX-map2 | 0 | 0 | F3 | 4B | | !3 | TILESTORED | sibmem, tmm1 |
| VEX-map3 | 0 | 0 | F2 | F0 | | | RORX | r32_r/r64_r, m32/m64/r32_b/r64_b, i8 |

## 3.2 NOTATIONAL CONVENTIONS

In the “Encoding/Instruction” descriptions of the EVEX map 4 instructions:

- “LLZ” means that the LL bits in the EVEX payload must be 0b00 and either bit being nonzero triggers #UD.
- “IGNORED” means that the W bit in the EVEX payload is ignored.
- “SCALABLE” means that the OSIZE of the instruction is variable and can be 64b, 32b or 16b. The OSIZE is determined by the W and pp bits in the EVEX payload as follows:
  - If W = 1, then OSIZE = 64b.
  - If W = 0 and pp = NP, then OSIZE = 32b.
  - If W = 0 and pp = 66, then OSIZE = 16b.

  The pp bits can only be NP or 66 for such instructions.
- “id”, “iw” and “ib” denotes an immediate of size 32b, 16b and 8b, respectively.
- “{NF}” means that the EVEX.NF bit is used to control status flags update: EVEX.NF = 1 (respectively, EVEX.NF = 0) suppresses (does not suppress) the status flags update.
- “{ND=ZU}” means that the EVEX.ND bit is used to control zero-upper behavior: EVEX.ND = 1 (respectively, EVEX.ND = 0) zero-uppers (does not zero-upper) the destination register.
- When EVEX.ND = 1/0 is used to signify the presence/absence of an NDD, there will be two separate encoding descriptions for the “{ND=0}” and “{ND=1}” cases.

# Chapter 4

# EXCEPTION CLASSES

## 4.1 EXCEPTION CLASS INSTRUCTION SUMMARY

| | | | |
|---|---|---|---|
| **Type AMX-E1-EVEX** | | | |
| LDTILECFG | m512 | APX-F-AMX | EVEX |
| **Type AMX-E2-EVEX** | | | |
| STTILECFG | m512 | APX-F-AMX | EVEX |
| **Type AMX-E3-EVEX** | | | |
| TILELOADD | tmm1, sibmem | APX-F-AMX | EVEX |
| TILELOADDT1 | tmm1, sibmem | APX-F-AMX | EVEX |
| TILESTORED | sibmem, tmm1 | APX-F-AMX | EVEX |
| **Type APX-EVEX-BMI** | | | |
| ANDN | r32, r32, m32/r32 | APX_F | EVEX |
| ANDN | r64, r64, m64/r64 | APX_F | EVEX |
| BEXTR | r32, m32/r32, r32 | APX_F | EVEX |
| BEXTR | r64, m64/r64, r64 | APX_F | EVEX |
| BLSI | r32, m32/r32 | APX_F | EVEX |
| BLSI | r64, m64/r64 | APX_F | EVEX |
| BLSMSK | r32, m32/r32 | APX_F | EVEX |
| BLSMSK | r64, m64/r64 | APX_F | EVEX |
| BLSR | r32, m32/r32 | APX_F | EVEX |
| BLSR | r64, m64/r64 | APX_F | EVEX |
| BZHI | r32, m32/r32, r32 | APX_F | EVEX |
| BZHI | r64, m64/r64, r64 | APX_F | EVEX |
| MULX | r32, r32, m32/r32, <edx:r:supp> | APX_F | EVEX |
| MULX | r64, r64, m64/r64, <rdx:r:supp> | APX_F | EVEX |
| PDEP | r32, r32, m32/r32 | APX_F | EVEX |
| PDEP | r64, r64, m64/r64 | APX_F | EVEX |
| PEXT | r32, r32, m32/r32 | APX_F | EVEX |
| PEXT | r64, r64, m64/r64 | APX_F | EVEX |
| RORX | r32, m32/r32, imm8 | APX_F | EVEX |
| RORX | r64, m64/r64, imm8 | APX_F | EVEX |
| SARX | r32, m32/r32, r32 | APX_F | EVEX |
| SARX | r64, m64/r64, r64 | APX_F | EVEX |
| SHLX | r32, m32/r32, r32 | APX_F | EVEX |
| SHLX | r64, m64/r64, r64 | APX_F | EVEX |
| SHRX | r32, m32/r32, r32 | APX_F | EVEX |
| SHRX | r64, m64/r64, r64 | APX_F | EVEX |
| **Type APX-EVEX-CCMP** | | | |
| CCMPB | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPB | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPB | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPB | rv, rv/mv, dfv | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-CCMP** *(contd.)* | | | |
|---|---|---|---|
| CCMPB | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPB | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPB | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPBE | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPBE | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPBE | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPBE | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPBE | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPBE | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPBE | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPF | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPF | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPF | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPF | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPF | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPF | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPF | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPL | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPL | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPL | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPL | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPL | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPL | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPL | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPLE | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPLE | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPLE | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPLE | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPLE | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPLE | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPLE | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPNB | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPNB | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPNB | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPNB | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPNB | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPNB | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPNB | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPNBE | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPNBE | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPNBE | r8/m8, r8, dfv | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-CCMP** *(contd.)* | | | |
|---|---|---|---|
| CCMPNBE | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPNBE | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPNBE | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPNBE | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPNL | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPNL | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPNL | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPNL | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPNL | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPNL | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPNL | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPNLE | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPNLE | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPNLE | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPNLE | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPNLE | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPNLE | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPNLE | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPNO | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPNO | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPNO | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPNO | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPNO | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPNO | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPNO | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPNS | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPNS | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPNS | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPNS | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPNS | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPNS | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPNS | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPNZ | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPNZ | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPNZ | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPNZ | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPNZ | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPNZ | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPNZ | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPO | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPO | r8/m8, imm8, dfv | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-CCMP** *(contd.)* | | | |
|---|---|---|---|
| CCMPO | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPO | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPO | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPO | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPO | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPS | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPS | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPS | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPS | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPS | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPS | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPS | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPT | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPT | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPT | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPT | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPT | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPT | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPT | rv/mv, rv, dfv | APX_F | EVEX |
| CCMPZ | r8, r8/m8, dfv | APX_F | EVEX |
| CCMPZ | r8/m8, imm8, dfv | APX_F | EVEX |
| CCMPZ | r8/m8, r8, dfv | APX_F | EVEX |
| CCMPZ | rv, rv/mv, dfv | APX_F | EVEX |
| CCMPZ | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CCMPZ | rv/mv, imm8, dfv | APX_F | EVEX |
| CCMPZ | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTB | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTB | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTB | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTB | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTBE | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTBE | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTBE | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTBE | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTF | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTF | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTF | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTF | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTL | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTL | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTL | rv/mv, imm16/imm32, dfv | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-CCMP** *(contd.)* | | | |
|---|---|---|---|
| CTESTL | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTLE | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTLE | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTLE | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTLE | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTNB | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTNB | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTNB | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTNB | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTNBE | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTNBE | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTNBE | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTNBE | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTNL | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTNL | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTNL | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTNL | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTNLE | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTNLE | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTNLE | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTNLE | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTNO | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTNO | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTNO | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTNO | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTNS | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTNS | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTNS | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTNS | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTNZ | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTNZ | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTNZ | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTNZ | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTO | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTO | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTO | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTO | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTS | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTS | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTS | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTS | rv/mv, rv, dfv | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-CCMP** *(contd.)* | | | |
|---|---|---|---|
| CTESTT | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTT | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTT | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTT | rv/mv, rv, dfv | APX_F | EVEX |
| CTESTZ | r8/m8, imm8, dfv | APX_F | EVEX |
| CTESTZ | r8/m8, r8, dfv | APX_F | EVEX |
| CTESTZ | rv/mv, imm16/imm32, dfv | APX_F | EVEX |
| CTESTZ | rv/mv, rv, dfv | APX_F | EVEX |
| **Type APX-EVEX-CET-WRSS** | | | |
| WRSSD | m32, r32 | APX_F | EVEX |
| WRSSQ | m64, r64 | APX_F | EVEX |
| **Type APX-EVEX-CET-WRUSS** | | | |
| WRUSSD | m32, r32 | APX_F | EVEX |
| WRUSSQ | m64, r64 | APX_F | EVEX |
| **Type APX-EVEX-CFCMOV** | | | |
| CFCMOVB | mv, rv | APX_F | EVEX |
| CFCMOVB | rv, rv | APX_F | EVEX |
| CFCMOVB | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVB | rv, rv/mv | APX_F | EVEX |
| CFCMOVBE | mv, rv | APX_F | EVEX |
| CFCMOVBE | rv, rv | APX_F | EVEX |
| CFCMOVBE | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVBE | rv, rv/mv | APX_F | EVEX |
| CFCMOVL | mv, rv | APX_F | EVEX |
| CFCMOVL | rv, rv | APX_F | EVEX |
| CFCMOVL | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVL | rv, rv/mv | APX_F | EVEX |
| CFCMOVLE | mv, rv | APX_F | EVEX |
| CFCMOVLE | rv, rv | APX_F | EVEX |
| CFCMOVLE | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVLE | rv, rv/mv | APX_F | EVEX |
| CFCMOVNB | mv, rv | APX_F | EVEX |
| CFCMOVNB | rv, rv | APX_F | EVEX |
| CFCMOVNB | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVNB | rv, rv/mv | APX_F | EVEX |
| CFCMOVNBE | mv, rv | APX_F | EVEX |
| CFCMOVNBE | rv, rv | APX_F | EVEX |
| CFCMOVNBE | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVNBE | rv, rv/mv | APX_F | EVEX |
| CFCMOVNL | mv, rv | APX_F | EVEX |
| CFCMOVNL | rv, rv | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-CFCMOV** *(contd.)* | | | |
|---|---|---|---|
| CFCMOVNL | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVNL | rv, rv/mv | APX_F | EVEX |
| CFCMOVNLE | mv, rv | APX_F | EVEX |
| CFCMOVNLE | rv, rv | APX_F | EVEX |
| CFCMOVNLE | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVNLE | rv, rv/mv | APX_F | EVEX |
| CFCMOVNO | mv, rv | APX_F | EVEX |
| CFCMOVNO | rv, rv | APX_F | EVEX |
| CFCMOVNO | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVNO | rv, rv/mv | APX_F | EVEX |
| CFCMOVNP | mv, rv | APX_F | EVEX |
| CFCMOVNP | rv, rv | APX_F | EVEX |
| CFCMOVNP | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVNP | rv, rv/mv | APX_F | EVEX |
| CFCMOVNS | mv, rv | APX_F | EVEX |
| CFCMOVNS | rv, rv | APX_F | EVEX |
| CFCMOVNS | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVNS | rv, rv/mv | APX_F | EVEX |
| CFCMOVNZ | mv, rv | APX_F | EVEX |
| CFCMOVNZ | rv, rv | APX_F | EVEX |
| CFCMOVNZ | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVNZ | rv, rv/mv | APX_F | EVEX |
| CFCMOVO | mv, rv | APX_F | EVEX |
| CFCMOVO | rv, rv | APX_F | EVEX |
| CFCMOVO | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVO | rv, rv/mv | APX_F | EVEX |
| CFCMOVP | mv, rv | APX_F | EVEX |
| CFCMOVP | rv, rv | APX_F | EVEX |
| CFCMOVP | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVP | rv, rv/mv | APX_F | EVEX |
| CFCMOVS | mv, rv | APX_F | EVEX |
| CFCMOVS | rv, rv | APX_F | EVEX |
| CFCMOVS | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVS | rv, rv/mv | APX_F | EVEX |
| CFCMOVZ | mv, rv | APX_F | EVEX |
| CFCMOVZ | rv, rv | APX_F | EVEX |
| CFCMOVZ | rv, rv, rv/mv | APX_F | EVEX |
| CFCMOVZ | rv, rv/mv | APX_F | EVEX |
| **Type APX-EVEX-CMPCCXADD** | | | |
| CMPBEXADD | m32, r32, r32 | APX_F | EVEX |
| CMPBEXADD | m64, r64, r64 | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-CMPCCXADD** *(contd.)* | | | |
|---|---|---|---|
| CMPBXADD | m32, r32, r32 | APX_F | EVEX |
| CMPBXADD | m64, r64, r64 | APX_F | EVEX |
| CMPLEXADD | m32, r32, r32 | APX_F | EVEX |
| CMPLEXADD | m64, r64, r64 | APX_F | EVEX |
| CMPLXADD | m32, r32, r32 | APX_F | EVEX |
| CMPLXADD | m64, r64, r64 | APX_F | EVEX |
| CMPNBEXADD | m32, r32, r32 | APX_F | EVEX |
| CMPNBEXADD | m64, r64, r64 | APX_F | EVEX |
| CMPNBXADD | m32, r32, r32 | APX_F | EVEX |
| CMPNBXADD | m64, r64, r64 | APX_F | EVEX |
| CMPNLEXADD | m32, r32, r32 | APX_F | EVEX |
| CMPNLEXADD | m64, r64, r64 | APX_F | EVEX |
| CMPNLXADD | m32, r32, r32 | APX_F | EVEX |
| CMPNLXADD | m64, r64, r64 | APX_F | EVEX |
| CMPNOXADD | m32, r32, r32 | APX_F | EVEX |
| CMPNOXADD | m64, r64, r64 | APX_F | EVEX |
| CMPNPXADD | m32, r32, r32 | APX_F | EVEX |
| CMPNPXADD | m64, r64, r64 | APX_F | EVEX |
| CMPNSXADD | m32, r32, r32 | APX_F | EVEX |
| CMPNSXADD | m64, r64, r64 | APX_F | EVEX |
| CMPNZXADD | m32, r32, r32 | APX_F | EVEX |
| CMPNZXADD | m64, r64, r64 | APX_F | EVEX |
| CMPOXADD | m32, r32, r32 | APX_F | EVEX |
| CMPOXADD | m64, r64, r64 | APX_F | EVEX |
| CMPPXADD | m32, r32, r32 | APX_F | EVEX |
| CMPPXADD | m64, r64, r64 | APX_F | EVEX |
| CMPSXADD | m32, r32, r32 | APX_F | EVEX |
| CMPSXADD | m64, r64, r64 | APX_F | EVEX |
| CMPZXADD | m32, r32, r32 | APX_F | EVEX |
| CMPZXADD | m64, r64, r64 | APX_F | EVEX |
| **Type APX-EVEX-ENQCMD** | | | |
| ENQCMD | ra, m512 | APX_F | EVEX |
| ENQCMDS | ra, m512 | APX_F | EVEX |
| **Type APX-EVEX-INT** | | | |
| ADC | r8, r8, r8/m8 | APX_F | EVEX |
| ADC | r8, r8/m8 | APX_F | EVEX |
| ADC | r8, r8/m8, imm8 | APX_F | EVEX |
| ADC | r8, r8/m8, r8 | APX_F | EVEX |
| ADC | r8/m8, imm8 | APX_F | EVEX |
| ADC | r8/m8, r8 | APX_F | EVEX |
| ADC | rv, rv, rv/mv | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-INT** *(contd.)* | | | |
|---|---|---|---|
| ADC | rv, rv/mv | APX_F | EVEX |
| ADC | rv, rv/mv, imm16/imm32 | APX_F | EVEX |
| ADC | rv, rv/mv, imm8 | APX_F | EVEX |
| ADC | rv, rv/mv, rv | APX_F | EVEX |
| ADC | rv/mv, imm16/imm32 | APX_F | EVEX |
| ADC | rv/mv, imm8 | APX_F | EVEX |
| ADC | rv/mv, rv | APX_F | EVEX |
| ADCX | r32, r32, r32/m32 | APX_F | EVEX |
| ADCX | r32, r32/m32 | APX_F | EVEX |
| ADCX | r64, r64, r64/m64 | APX_F | EVEX |
| ADCX | r64, r64/m64 | APX_F | EVEX |
| ADD | r8, r8, r8/m8 | APX_F | EVEX |
| ADD | r8, r8/m8 | APX_F | EVEX |
| ADD | r8, r8/m8, imm8 | APX_F | EVEX |
| ADD | r8, r8/m8, r8 | APX_F | EVEX |
| ADD | r8/m8, imm8 | APX_F | EVEX |
| ADD | r8/m8, r8 | APX_F | EVEX |
| ADD | rv, rv, rv/mv | APX_F | EVEX |
| ADD | rv, rv/mv | APX_F | EVEX |
| ADD | rv, rv/mv, imm16/imm32 | APX_F | EVEX |
| ADD | rv, rv/mv, imm8 | APX_F | EVEX |
| ADD | rv, rv/mv, rv | APX_F | EVEX |
| ADD | rv/mv, imm16/imm32 | APX_F | EVEX |
| ADD | rv/mv, imm8 | APX_F | EVEX |
| ADD | rv/mv, rv | APX_F | EVEX |
| ADOX | r32, r32, r32/m32 | APX_F | EVEX |
| ADOX | r32, r32/m32 | APX_F | EVEX |
| ADOX | r64, r64, r64/m64 | APX_F | EVEX |
| ADOX | r64, r64/m64 | APX_F | EVEX |
| AND | r8, r8, r8/m8 | APX_F | EVEX |
| AND | r8, r8/m8 | APX_F | EVEX |
| AND | r8, r8/m8, imm8 | APX_F | EVEX |
| AND | r8, r8/m8, r8 | APX_F | EVEX |
| AND | r8/m8, imm8 | APX_F | EVEX |
| AND | r8/m8, r8 | APX_F | EVEX |
| AND | rv, rv, rv/mv | APX_F | EVEX |
| AND | rv, rv/mv | APX_F | EVEX |
| AND | rv, rv/mv, imm16/imm32 | APX_F | EVEX |
| AND | rv, rv/mv, imm8 | APX_F | EVEX |
| AND | rv, rv/mv, rv | APX_F | EVEX |
| AND | rv/mv, imm16/imm32 | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-INT** *(contd.)* | | | |
|---|---|---|---|
| AND | rv/mv, imm8 | APX_F | EVEX |
| AND | rv/mv, rv | APX_F | EVEX |
| CMOVB | rv, rv, rv/mv | APX_F | EVEX |
| CMOVBE | rv, rv, rv/mv | APX_F | EVEX |
| CMOVL | rv, rv, rv/mv | APX_F | EVEX |
| CMOVLE | rv, rv, rv/mv | APX_F | EVEX |
| CMOVNB | rv, rv, rv/mv | APX_F | EVEX |
| CMOVNBE | rv, rv, rv/mv | APX_F | EVEX |
| CMOVNL | rv, rv, rv/mv | APX_F | EVEX |
| CMOVNLE | rv, rv, rv/mv | APX_F | EVEX |
| CMOVNO | rv, rv, rv/mv | APX_F | EVEX |
| CMOVNP | rv, rv, rv/mv | APX_F | EVEX |
| CMOVNS | rv, rv, rv/mv | APX_F | EVEX |
| CMOVNZ | rv, rv, rv/mv | APX_F | EVEX |
| CMOVO | rv, rv, rv/mv | APX_F | EVEX |
| CMOVP | rv, rv, rv/mv | APX_F | EVEX |
| CMOVS | rv, rv, rv/mv | APX_F | EVEX |
| CMOVZ | rv, rv, rv/mv | APX_F | EVEX |
| CRC32 | ry, r8/m8 | APX_F | EVEX |
| CRC32 | ry, rv/mv | APX_F | EVEX |
| DEC | r8, r8/m8 | APX_F | EVEX |
| DEC | r8/m8 | APX_F | EVEX |
| DEC | rv, rv/mv | APX_F | EVEX |
| DEC | rv/mv | APX_F | EVEX |
| DIV | r8/m8, <ax:rw:supp> | APX_F | EVEX |
| DIV | rv/mv, <orax:rw:supp>, <ordx:rw:supp> | APX_F | EVEX |
| IDIV | r8/m8, <ax:rw:supp> | APX_F | EVEX |
| IDIV | rv/mv, <orax:rw:supp>, <ordx:rw:supp> | APX_F | EVEX |
| IMUL | r8/m8, <al:r:supp>, <ax:w:supp> | APX_F | EVEX |
| IMUL | rv, rv, rv/mv | APX_F | EVEX |
| IMUL | rv, rv/mv | APX_F | EVEX |
| IMUL | rv, rv/mv, imm16/imm32 | APX_F | EVEX |
| IMUL | rv, rv/mv, imm8 | APX_F | EVEX |
| IMUL | rv/mv, <orax:rw:supp>, <ordx:w:supp> | APX_F | EVEX |
| INC | r8, r8/m8 | APX_F | EVEX |
| INC | r8/m8 | APX_F | EVEX |
| INC | rv, rv/mv | APX_F | EVEX |
| INC | rv/mv | APX_F | EVEX |
| LZCNT | rv, rv/mv | APX_F | EVEX |
| MOVBE | rv, rv/mv | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-INT** *(contd.)* | | | |
|---|---|---|---|
| MOVBE | rv/mv, rv | APX_F | EVEX |
| MOVDIR64B | ra, m512 | APX_F | EVEX |
| MOVDIRI | my, ry | APX_F | EVEX |
| MUL | r8/m8, <al:r:supp>, <ax:w:supp> | APX_F | EVEX |
| MUL | rv/mv, <orax:rw:supp>, <ordx:w:supp> | APX_F | EVEX |
| NEG | r8, r8/m8 | APX_F | EVEX |
| NEG | r8/m8 | APX_F | EVEX |
| NEG | rv, rv/mv | APX_F | EVEX |
| NEG | rv/mv | APX_F | EVEX |
| NOT | r8, r8/m8 | APX_F | EVEX |
| NOT | r8/m8 | APX_F | EVEX |
| NOT | rv, rv/mv | APX_F | EVEX |
| NOT | rv/mv | APX_F | EVEX |
| OR | r8, r8, r8/m8 | APX_F | EVEX |
| OR | r8, r8/m8 | APX_F | EVEX |
| OR | r8, r8/m8, imm8 | APX_F | EVEX |
| OR | r8, r8/m8, r8 | APX_F | EVEX |
| OR | r8/m8, imm8 | APX_F | EVEX |
| OR | r8/m8, r8 | APX_F | EVEX |
| OR | rv, rv, rv/mv | APX_F | EVEX |
| OR | rv, rv/mv | APX_F | EVEX |
| OR | rv, rv/mv, imm16/imm32 | APX_F | EVEX |
| OR | rv, rv/mv, imm8 | APX_F | EVEX |
| OR | rv, rv/mv, rv | APX_F | EVEX |
| OR | rv/mv, imm16/imm32 | APX_F | EVEX |
| OR | rv/mv, imm8 | APX_F | EVEX |
| OR | rv/mv, rv | APX_F | EVEX |
| POPCNT | rv, rv/mv | APX_F | EVEX |
| RCL | r8, r8/m8, <1:r:impl> | APX_F | EVEX |
| RCL | r8, r8/m8, <cl:r:impl> | APX_F | EVEX |
| RCL | r8, r8/m8, imm8 | APX_F | EVEX |
| RCL | r8/m8, <1:r:impl> | APX_F | EVEX |
| RCL | r8/m8, <cl:r:impl> | APX_F | EVEX |
| RCL | r8/m8, imm8 | APX_F | EVEX |
| RCL | rv, rv/mv, <1:r:impl> | APX_F | EVEX |
| RCL | rv, rv/mv, <cl:r:impl> | APX_F | EVEX |
| RCL | rv, rv/mv, imm8 | APX_F | EVEX |
| RCL | rv/mv, <1:r:impl> | APX_F | EVEX |
| RCL | rv/mv, <cl:r:impl> | APX_F | EVEX |
| RCL | rv/mv, imm8 | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-INT** *(contd.)* | | | |
|---|---|---|---|
| RCR | r8, r8/m8, <1:r:impl> | APX_F | EVEX |
| RCR | r8, r8/m8, <cl:r:impl> | APX_F | EVEX |
| RCR | r8, r8/m8, imm8 | APX_F | EVEX |
| RCR | r8/m8, <1:r:impl> | APX_F | EVEX |
| RCR | r8/m8, <cl:r:impl> | APX_F | EVEX |
| RCR | r8/m8, imm8 | APX_F | EVEX |
| RCR | rv, rv/mv, <1:r:impl> | APX_F | EVEX |
| RCR | rv, rv/mv, <cl:r:impl> | APX_F | EVEX |
| RCR | rv, rv/mv, imm8 | APX_F | EVEX |
| RCR | rv/mv, <1:r:impl> | APX_F | EVEX |
| RCR | rv/mv, <cl:r:impl> | APX_F | EVEX |
| RCR | rv/mv, imm8 | APX_F | EVEX |
| ROL | r8, r8/m8, <1:r:impl> | APX_F | EVEX |
| ROL | r8, r8/m8, <cl:r:impl> | APX_F | EVEX |
| ROL | r8, r8/m8, imm8 | APX_F | EVEX |
| ROL | r8/m8, <1:r:impl> | APX_F | EVEX |
| ROL | r8/m8, <cl:r:impl> | APX_F | EVEX |
| ROL | r8/m8, imm8 | APX_F | EVEX |
| ROL | rv, rv/mv, <1:r:impl> | APX_F | EVEX |
| ROL | rv, rv/mv, <cl:r:impl> | APX_F | EVEX |
| ROL | rv, rv/mv, imm8 | APX_F | EVEX |
| ROL | rv/mv, <1:r:impl> | APX_F | EVEX |
| ROL | rv/mv, <cl:r:impl> | APX_F | EVEX |
| ROL | rv/mv, imm8 | APX_F | EVEX |
| ROR | r8, r8/m8, <1:r:impl> | APX_F | EVEX |
| ROR | r8, r8/m8, <cl:r:impl> | APX_F | EVEX |
| ROR | r8, r8/m8, imm8 | APX_F | EVEX |
| ROR | r8/m8, <1:r:impl> | APX_F | EVEX |
| ROR | r8/m8, <cl:r:impl> | APX_F | EVEX |
| ROR | r8/m8, imm8 | APX_F | EVEX |
| ROR | rv, rv/mv, <1:r:impl> | APX_F | EVEX |
| ROR | rv, rv/mv, <cl:r:impl> | APX_F | EVEX |
| ROR | rv, rv/mv, imm8 | APX_F | EVEX |
| ROR | rv/mv, <1:r:impl> | APX_F | EVEX |
| ROR | rv/mv, <cl:r:impl> | APX_F | EVEX |
| ROR | rv/mv, imm8 | APX_F | EVEX |
| SAR | r8, r8/m8, <1:r:impl> | APX_F | EVEX |
| SAR | r8, r8/m8, <cl:r:impl> | APX_F | EVEX |
| SAR | r8, r8/m8, imm8 | APX_F | EVEX |
| SAR | r8/m8, <1:r:impl> | APX_F | EVEX |
| SAR | r8/m8, <cl:r:impl> | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-INT** *(contd.)* | | | |
|---|---|---|---|
| SAR | r8/m8, imm8 | APX_F | EVEX |
| SAR | rv, rv/mv, <1:r:impl> | APX_F | EVEX |
| SAR | rv, rv/mv, <cl:r:impl> | APX_F | EVEX |
| SAR | rv, rv/mv, imm8 | APX_F | EVEX |
| SAR | rv/mv, <1:r:impl> | APX_F | EVEX |
| SAR | rv/mv, <cl:r:impl> | APX_F | EVEX |
| SAR | rv/mv, imm8 | APX_F | EVEX |
| SBB | r8, r8, r8/m8 | APX_F | EVEX |
| SBB | r8, r8/m8 | APX_F | EVEX |
| SBB | r8, r8/m8, imm8 | APX_F | EVEX |
| SBB | r8, r8/m8, r8 | APX_F | EVEX |
| SBB | r8/m8, imm8 | APX_F | EVEX |
| SBB | r8/m8, r8 | APX_F | EVEX |
| SBB | rv, rv, rv/mv | APX_F | EVEX |
| SBB | rv, rv/mv | APX_F | EVEX |
| SBB | rv, rv/mv, imm16/imm32 | APX_F | EVEX |
| SBB | rv, rv/mv, imm8 | APX_F | EVEX |
| SBB | rv, rv/mv, rv | APX_F | EVEX |
| SBB | rv/mv, imm16/imm32 | APX_F | EVEX |
| SBB | rv/mv, imm8 | APX_F | EVEX |
| SBB | rv/mv, rv | APX_F | EVEX |
| SETB | r8/m8 | APX_F | EVEX |
| SETBE | r8/m8 | APX_F | EVEX |
| SETL | r8/m8 | APX_F | EVEX |
| SETLE | r8/m8 | APX_F | EVEX |
| SETNB | r8/m8 | APX_F | EVEX |
| SETNBE | r8/m8 | APX_F | EVEX |
| SETNL | r8/m8 | APX_F | EVEX |
| SETNLE | r8/m8 | APX_F | EVEX |
| SETNO | r8/m8 | APX_F | EVEX |
| SETNP | r8/m8 | APX_F | EVEX |
| SETNS | r8/m8 | APX_F | EVEX |
| SETNZ | r8/m8 | APX_F | EVEX |
| SETO | r8/m8 | APX_F | EVEX |
| SETP | r8/m8 | APX_F | EVEX |
| SETS | r8/m8 | APX_F | EVEX |
| SETZ | r8/m8 | APX_F | EVEX |
| SHL | r8, r8/m8, <1:r:impl> | APX_F | EVEX |
| SHL | r8, r8/m8, <cl:r:impl> | APX_F | EVEX |
| SHL | r8, r8/m8, imm8 | APX_F | EVEX |
| SHL | r8/m8, <1:r:impl> | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-INT** *(contd.)* | | | |
|---|---|---|---|
| SHL | r8/m8, <cl:r:impl> | APX_F | EVEX |
| SHL | r8/m8, imm8 | APX_F | EVEX |
| SHL | rv, rv/mv, <1:r:impl> | APX_F | EVEX |
| SHL | rv, rv/mv, <cl:r:impl> | APX_F | EVEX |
| SHL | rv, rv/mv, imm8 | APX_F | EVEX |
| SHL | rv/mv, <1:r:impl> | APX_F | EVEX |
| SHL | rv/mv, <cl:r:impl> | APX_F | EVEX |
| SHL | rv/mv, imm8 | APX_F | EVEX |
| SHLD | rv, rv/mv, rv, <cl:r:impl> | APX_F | EVEX |
| SHLD | rv, rv/mv, rv, imm8 | APX_F | EVEX |
| SHLD | rv/mv, rv, <cl:r:impl> | APX_F | EVEX |
| SHLD | rv/mv, rv, imm8 | APX_F | EVEX |
| SHR | r8, r8/m8, <1:r:impl> | APX_F | EVEX |
| SHR | r8, r8/m8, <cl:r:impl> | APX_F | EVEX |
| SHR | r8, r8/m8, imm8 | APX_F | EVEX |
| SHR | r8/m8, <1:r:impl> | APX_F | EVEX |
| SHR | r8/m8, <cl:r:impl> | APX_F | EVEX |
| SHR | r8/m8, imm8 | APX_F | EVEX |
| SHR | rv, rv/mv, <1:r:impl> | APX_F | EVEX |
| SHR | rv, rv/mv, <cl:r:impl> | APX_F | EVEX |
| SHR | rv, rv/mv, imm8 | APX_F | EVEX |
| SHR | rv/mv, <1:r:impl> | APX_F | EVEX |
| SHR | rv/mv, <cl:r:impl> | APX_F | EVEX |
| SHR | rv/mv, imm8 | APX_F | EVEX |
| SHRD | rv, rv/mv, rv, <cl:r:impl> | APX_F | EVEX |
| SHRD | rv, rv/mv, rv, imm8 | APX_F | EVEX |
| SHRD | rv/mv, rv, <cl:r:impl> | APX_F | EVEX |
| SHRD | rv/mv, rv, imm8 | APX_F | EVEX |
| SUB | r8, r8, r8/m8 | APX_F | EVEX |
| SUB | r8, r8/m8 | APX_F | EVEX |
| SUB | r8, r8/m8, imm8 | APX_F | EVEX |
| SUB | r8, r8/m8, r8 | APX_F | EVEX |
| SUB | r8/m8, imm8 | APX_F | EVEX |
| SUB | r8/m8, r8 | APX_F | EVEX |
| SUB | rv, rv, rv/mv | APX_F | EVEX |
| SUB | rv, rv/mv | APX_F | EVEX |
| SUB | rv, rv/mv, imm16/imm32 | APX_F | EVEX |
| SUB | rv, rv/mv, imm8 | APX_F | EVEX |
| SUB | rv, rv/mv, rv | APX_F | EVEX |
| SUB | rv/mv, imm16/imm32 | APX_F | EVEX |
| SUB | rv/mv, imm8 | APX_F | EVEX |

*Table continued on next page...*

| | | | |
|---|---|---|---|
| **Type APX-EVEX-INT** *(contd.)* | | | |
| SUB | rv/mv, rv | APX_F | EVEX |
| TZCNT | rv, rv/mv | APX_F | EVEX |
| XOR | r8, r8, r8/m8 | APX_F | EVEX |
| XOR | r8, r8/m8 | APX_F | EVEX |
| XOR | r8, r8/m8, imm8 | APX_F | EVEX |
| XOR | r8, r8/m8, r8 | APX_F | EVEX |
| XOR | r8/m8, imm8 | APX_F | EVEX |
| XOR | r8/m8, r8 | APX_F | EVEX |
| XOR | rv, rv, rv/mv | APX_F | EVEX |
| XOR | rv, rv/mv | APX_F | EVEX |
| XOR | rv, rv/mv, imm16/imm32 | APX_F | EVEX |
| XOR | rv, rv/mv, imm8 | APX_F | EVEX |
| XOR | rv, rv/mv, rv | APX_F | EVEX |
| XOR | rv/mv, imm16/imm32 | APX_F | EVEX |
| XOR | rv/mv, imm8 | APX_F | EVEX |
| XOR | rv/mv, rv | APX_F | EVEX |
| **Type APX-EVEX-INVEPT** | | | |
| INVEPT | r64, m128 | APX_F | EVEX |
| **Type APX-EVEX-INVPCID** | | | |
| INVPCID | r64, m128 | APX_F | EVEX |
| **Type APX-EVEX-INVVPID** | | | |
| INVVPID | r64, m128 | APX_F | EVEX |
| **Type APX-EVEX-KEYLOCKER** | | | |
| AESDEC128KL | xmm1, m384 | APX_F | EVEX |
| AESDEC256KL | xmm1, m512 | APX_F | EVEX |
| AESDECWIDE128KL | m384, <xmm0:rw>, <xmm1:rw>, <xmm2:rw>, <xmm3:rw>, <xmm4:rw>, <xmm5:rw>, <xmm6:rw>, <xmm7:rw> | APX_F | EVEX |
| AESDECWIDE256KL | m512, <xmm0:rw>, <xmm1:rw>, <xmm2:rw>, <xmm3:rw>, <xmm4:rw>, <xmm5:rw>, <xmm6:rw>, <xmm7:rw> | APX_F | EVEX |
| AESENC128KL | xmm1, m384 | APX_F | EVEX |
| AESENC256KL | xmm1, m512 | APX_F | EVEX |
| AESENCWIDE128KL | m384, <xmm0:rw>, <xmm1:rw>, <xmm2:rw>, <xmm3:rw>, <xmm4:rw>, <xmm5:rw>, <xmm6:rw>, <xmm7:rw> | APX_F | EVEX |
| AESENCWIDE256KL | m512, <xmm0:rw>, <xmm1:rw>, <xmm2:rw>, <xmm3:rw>, <xmm4:rw>, <xmm5:rw>, <xmm6:rw>, <xmm7:rw> | APX_F | EVEX |
| ENCODEKEY128 | r32, r32, <xmm0:rw>, <xmm1:w>, <xmm2:w>, <xmm4:w>, <xmm5:w>, <xmm6:w> | APX_F | EVEX |

*Table continued on next page...*

| **Type APX-EVEX-KEYLOCKER** *(contd.)* | | | |
|---|---|---|---|
| ENCODEKEY256 | r32, r32, <xmm0:rw>, <xmm1:rw>, <xmm2:w>, <xmm3:w>, <xmm4:w>, <xmm5:w>, <xmm6:w> | APX_F | EVEX |
| **Type APX-EVEX-KMOV** | | | |
| KMOVB | k1, k2/m8 | APX_F | EVEX |
| KMOVB | k1, r32 | APX_F | EVEX |
| KMOVB | m8, k1 | APX_F | EVEX |
| KMOVB | r32, k1 | APX_F | EVEX |
| KMOVD | k1, k2/m32 | APX_F | EVEX |
| KMOVD | k1, r32 | APX_F | EVEX |
| KMOVD | m32, k1 | APX_F | EVEX |
| KMOVD | r32, k1 | APX_F | EVEX |
| KMOVQ | k1, k2/m64 | APX_F | EVEX |
| KMOVQ | k1, r64 | APX_F | EVEX |
| KMOVQ | m64, k1 | APX_F | EVEX |
| KMOVQ | r64, k1 | APX_F | EVEX |
| KMOVW | k1, k2/m16 | APX_F | EVEX |
| KMOVW | k1, r32 | APX_F | EVEX |
| KMOVW | m16, k1 | APX_F | EVEX |
| KMOVW | r32, k1 | APX_F | EVEX |
| **Type APX-EVEX-PP2** | | | |
| POP2 | r64, r64, <pop:rw:supp> | APX_F | EVEX |
| POP2P | r64, r64, <pop:rw:supp> | APX_F | EVEX |
| PUSH2 | r64, r64, <push:rw:supp> | APX_F | EVEX |
| PUSH2P | r64, r64, <push:rw:supp> | APX_F | EVEX |
| **Type APX-EVEX-SHA** | | | |
| SHA1MSG1 | xmm1, xmm2/m128 | APX_F | EVEX |
| SHA1MSG2 | xmm1, xmm2/m128 | APX_F | EVEX |
| SHA1NEXTE | xmm1, xmm2/m128 | APX_F | EVEX |
| SHA1RNDS4 | xmm1, xmm2/m128, imm8 | APX_F | EVEX |
| SHA256MSG1 | xmm1, xmm2/m128 | APX_F | EVEX |
| SHA256MSG2 | xmm1, xmm2/m128 | APX_F | EVEX |
| SHA256RNDS2 | xmm1, xmm2/m128, <xmm0> | APX_F | EVEX |
| **Type APX-LEGACY-JMPABS** | | | |
| JMPABS | target64 | APX_F | LEGACY |

## 4.2 EXCEPTION CLASS SUMMARY

### 4.2.1 EXCEPTION CLASS AMX-E1-EVEX

| | |
|---|---|
| AMX-E1-EVEX | • All of AMX-E1<br>• #UD if EVEX.z != 0b0 // P2[7]<br>• #UD if EVEX.LL' != 0b00 // P2[6:5]<br>• #UD if EVEX.b != 0b0 // P2[4]<br>• #UD if EVEX.aaa != 0b000 // P2[2:0]<br>• #UD if EVEX.VVVV != 0b1111 // P1[6:3]<br>• #UD if EVEX.V' != 0b1 // P2[3] |

Table 4.2: Type AMX-E1-EVEX Class Exception Conditions

### 4.2.2 EXCEPTION CLASS AMX-E2-EVEX

| | |
|---|---|
| AMX-E2-EVEX | • All of AMX-E2<br>• #UD if EVEX.z != 0b0 // P2[7]<br>• #UD if EVEX.LL' != 0b00 // P2[6:5]<br>• #UD if EVEX.b != 0b0 // P2[4]<br>• #UD if EVEX.aaa != 0b000 // P2[2:0]<br>• #UD if EVEX.VVVV != 0b1111 // P1[6:3]<br>• #UD if EVEX.V' != 0b1 // P2[3] |

Table 4.4: Type AMX-E2-EVEX Class Exception Conditions

### 4.2.3 EXCEPTION CLASS AMX-E3-EVEX

| | |
|---|---|
| AMX-E3-EVEX | • All of AMX-E3<br>• #UD if EVEX.z != 0b0 // P2[7]<br>• #UD if EVEX.LL' != 0b00 // P2[6:5]<br>• #UD if EVEX.b != 0b0 // P2[4]<br>• #UD if EVEX.aaa != 0b000 // P2[2:0]<br>• #UD if EVEX.VVVV != 0b1111 // P1[6:3]<br>• #UD if EVEX.V' != 0b1 // P2[3] |

Table 4.6: Type AMX-E3-EVEX Class Exception Conditions

### 4.2.4 EXCEPTION CLASS APX-EVEX-BMI

This exception type is applicable to EVEX-encoded BMI instructions, which are promoted by Intel® APX from VEX space into EVEX space with the same map ids (maps 2 and 3) and opcodes and have the following mnemonics:

ANDN, BEXTR, BLSI, BLSMSK, BLSR, BZHI, MULX, PDEP, PEXT, RORX, SARX, SHLX, SHRX

Of them, the following support NF:

ANDN, BEXTR, BLSI, BLSMSK, BLSR, BZHI

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present with a BMI instruction's map id and opcode |
| | | | | X | EVEX prefix is present with a BMI instruction's map id and opcode, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present with a BMI instruction's map id and opcode and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than {V4,L,NF} is 1.<br>• EVEX.L=1.<br>• EVEX.NF=1 and the instruction does not support NF.<br>• Any #UD condition in SDM, vol.2, tables 2-37 and 2-39. |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If the APX_F or any instruction-specific CPUID feature flag is 0. |
| Stack, #SS(0) | | | | X | If a memory address referencing the SS segment is in a non-canonical form. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs. |
| Alignment Check #AC(0) | | | | X | If alignment checking is enabled and an unaligned memory access is made while CPL=3 |

Table 4.7: Type APX-EVEX-BMI Class Exception Conditions

### 4.2.5 EXCEPTION CLASS APX-EVEX-CCMP

This exception type is applicable to CCMP (Conditional CMP) and CTEST (Conditional TEST) instructions, which are new EVEX map 4 instructions introduced by APX.

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present and EVEX.map=4 |
| | | | | X | EVEX prefix is present and EVEX.map=4, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present and EVEX.map=4 and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than {SC3,SC2,SC1,SC0} is 1. |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If CPUID feature flag APX_F is 0. |
| Stack, #SS(0) | | | | X | If a memory address referencing the SS segment is in a non-canonical form. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs. |
| Alignment Check #AC(0) | | | | X | If alignment checking is enabled and an unaligned memory access is made while CPL=3 |

Table 4.8: Type APX-EVEX-CCMP Class Exception Conditions

### 4.2.6 EXCEPTION CLASS APX-EVEX-CET-WRSS

This exception type is applicable to EVEX-encoded CET instructions WRSSD and WRSSQ, which are promoted by Intel® APX from legacy space into EVEX map 4.

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present and EVEX.map=4 |
| | | | | X | EVEX prefix is present and EVEX.map=4, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present and EVEX.map=4 and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than V4 is 1.<br>• Any of EVEX.{V4,V3,V2,V1,V0} is 0.<br>• ModRM.Mod=3<br>• CR4.CET=0.<br>• CPL=3 and IA32_U_CET.SH_STK_EN=0.<br>• CPL<3 and IA32_S_CET.SH_STK_EN=0.<br>• CPL=3 and IA32_U_CET.WR_SHSTK_EN=0.<br>• CPL<3 and IA32_S_CET.WR_SHSTK_EN=0. |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If the APX_F or any instruction-specific CPUID feature flag is 0. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| | | | | X | If the memory address is not 4-byte aligned. |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs. |
| | | | | X | If CPL=3 and the destination is not a user shadow stack. |
| | | | | X | If CPL<3 and the destination is not a supervisor shadow stack. |
| | | | | X | Other terminal and non-terminal faults. |

Table 4.9: Type APX-EVEX-CET-WRSS Class Exception Conditions

### 4.2.7 EXCEPTION CLASS APX-EVEX-CET-WRUSS

This exception type is applicable to EVEX-encoded CET instructions WRUSSD and WRUSSQ, which are promoted by Intel® APX from legacy space into EVEX map 4.

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present and EVEX.map=4 |
| | | | | X | EVEX prefix is present and EVEX.map=4, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present and EVEX.map=4 and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than V4 is 1.<br>• Any of EVEX.{V4,V3,V2,V1,V0} is 0.<br>• ModRM.Mod=3<br>• CR4.CET=0. |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If the APX_F or any instruction-specific CPUID feature flag is 0. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| | | | | X | If the memory address is not 4-byte aligned. |
| | | | | X | If CPL$\neq$0. |
| Page Fault #PF(faultcode) | | | | X | If the destination is not a user shadow stack. |
| | | | | X | Other terminal and non-terminal faults. |

Table 4.10: Type APX-EVEX-CET-WRUSS Class Exception Conditions

### 4.2.8 EXCEPTION CLASS APX-EVEX-CFCMOV

This exception type is applicable to CFCMOVcc (Conditional Faulting CMOVcc) instructions, which are new EVEX map 4 instructions introduced by APX. When the condition code evaluates to false, a CFCMOVcc instruction suppresses all memory faults and the debug exception (#DB).

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present and EVEX.map=4 |
| | | | | X | EVEX prefix is present and EVEX.map=4, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present and EVEX.map=4 and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than {V4,ND,NF} is 1.<br>• EVEX.ND=0 and any of EVEX.{V4,V3,V2,V1,V0} is 0. |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If CPUID feature flag APX_F is 0. |
| Stack, #SS(0) | | | | X | If a memory address referencing the SS segment is in a non-canonical form while the condition code evaluates to true.. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form while the condition code evaluates to true.. |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs while the condition code evaluates to true. |
| Alignment Check #AC(0) | | | | X | If alignment checking is enabled and an unaligned memory access is made while CPL=3 and the condition code evaluates to true. |
| Debug Exception #DB | | | | X | If #DB is triggered while the condition code evaluates to true. |

Table 4.11: Type APX-EVEX-CFCMOV Class Exception Conditions

### 4.2.9 EXCEPTION CLASS APX-EVEX-CMPCCXADD

This exception type is applicable to EVEX-encoded CMPccXADD instructions, which are promoted by Intel® APX from VEX space into EVEX space with the same map (map 2) id and opcodes.

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present with a CMPccXADD instruction's map id and opcode |
| | | | | X | EVEX prefix is present with a CMPccXADD instruction's map id and opcode, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present with a CMPccXADD instruction's map id and opcode and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than {V4,L} is 1.<br>• EVEX.L=1.<br>• Any #UD condition in SDM, vol.2, tables 2-37 and 2-39. |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If the APX_F or any instruction-specific CPUID feature flag is 0. |
| Stack, #SS(0) | | | | X | If a memory address referencing the SS segment is in a non-canonical form. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| | | | | X | If the memory address is not naturally aligned (4/8 bytes). |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs. |

Table 4.12: Type APX-EVEX-CMPCCXADD Class Exception Conditions

### 4.2.10 EXCEPTION CLASS APX-EVEX-ENQCMD

This exception type is applicable to the EVEX-encoded ENQCMD and ENQCMDS instructions, which are promoted by Intel® APX from legacy space into EVEX map 4.

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present and EVEX.map=4 |
| | | | | X | EVEX prefix is present and EVEX.map=4, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present and EVEX.map=4 and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than V4 is 1.<br>• Any of EVEX.{V4,V3,V2,V1,V0} is 0.<br>• ModRM.Mod=3. |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If the APX_F or any instruction-specific CPUID feature flag is 0. |
| Stack, #SS(0) | | | | X | If a memory address referencing the SS segment is in a non-canonical form. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| | | | | X | If the memory address is not 64-byte aligned. |
| | | | | X | If bits 30:20 of the source operand are not all zero. |
| | | | | X | (For ENQCMD only) If bits 19:0 of the source operand are not all zero. |
| | | | | X | (For ENQCMD only) If IA32_PASID.PASID_Valid=0. |
| | | | | X | (For ENQCMDS only) If CPL$\neq$0. |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs. |

Table 4.13: Type APX-EVEX-ENQCMD Class Exception Conditions

### 4.2.11 EXCEPTION CLASS APX-EVEX-INT

This exception type is applicable to EVEX-encoded integer instructions which are promoted by Intel® APX from legacy space into EVEX map 4 and have the following mnemonics:

> ADC, ADCX, ADD, ADOX, AND, CMOVcc, CRC32, DEC, DIV, IDIV, IMUL, INC, LZCNT, MOVBE, MOVDIR64B, MOVDIRI, MUL, NEG, NOT, POPCNT, OR, RCL, RCR, ROL, ROR, SAR, SBB, SETcc, SHL, SHLD, SHR, SHRD, SUB, TZCNT, XOR

of which ADCX, ADOX, CRC32, MOVBE, POPCNT, MOVDIR64B, MOVDIRI also have instruction-specific CPUID feature flags.

The following EVEX map 4 instructions support NDD:

> INC, DEC, NOT, NEG, ADD, SUB, ADC, SBB, AND, OR, XOR, SAL, SAR, SHL, SHR, RCL, RCR, ROL, ROR, SHLD, SHRD, ADCX, ADOX, CMOVcc, and IMUL with opcode 0xAF

The following EVEX map 4 instructions support ZU:

> SETcc and IMUL with opcodes 0x69 and 0x6B

The following EVEX map 4 instructions support NF:

> INC, DEC, NEG, ADD, SUB, AND, OR, XOR, SAL, SAR, SHL, SHR, ROL, ROR, SHLD, SHRD, IMUL, IDIV, MUL, DIV, LZCNT, TZCNT, POPCNT

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present and EVEX.map=4 |
| | | | | X | EVEX prefix is present and EVEX.map=4, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present and EVEX.map=4 and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than {V4,ND,NF} is 1.<br>• EVEX.ND=0 and any of EVEX.{V4,V3,V2,V1,V0} is 0.<br>• EVEX.ND=1 and the instruction does not support NDD or ZU.<br>• Any of EVEX.{V4,V3,V2,V1,V0} is 0 and the instruction supports ZU.<br>• EVEX.NF=1 and the instruction does not support NF. |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If the APX_F or any instruction-specific CPUID feature flag is 0. |
| Stack, #SS(0) | | | | X | If a memory address referencing the SS segment is in a non-canonical form. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| | | | | X | (MOVDIR64B only) If the address in the destination (register) operand is not aligned to a 64-byte boundary. |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs. |
| Alignment Check #AC(0) | | | | X | If alignment checking is enabled and an unaligned memory access is made while CPL=3 |
| Divide Error #DE | | | | X | (DIV and IDIV only) If the source divisor is 0 or if the quotient is too large for the destination register. |

Table 4.14: Type APX-EVEX-INT Class Exception Conditions

### 4.2.12 EXCEPTION CLASS APX-EVEX-INVEPT

This exception type is applicable to the EVEX-encoded INVEPT instruction, which is promoted by Intel® APX from legacy space into EVEX map 4.

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present and EVEX.map=4 |
| | | | | X | EVEX prefix is present and EVEX.map=4, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present and EVEX.map=4 and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than V4 is 1.<br>• Any of EVEX.{V4,V3,V2,V1,V0} is 0.<br>• ModRM.Mod=3.<br>• Not in VMX operation.<br>• The logical processor does not support EPT.<br>• The logical processor supports EPT but does not support the INVEPT instruction. |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If the APX_F or any instruction-specific CPUID feature flag is 0. |
| Stack, #SS(0) | | | | X | If a memory address referencing the SS segment is in a non-canonical form. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| | | | | X | If CPL$\neq$0. |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs. |

Table 4.15: Type APX-EVEX-INVEPT Class Exception Conditions

### 4.2.13 EXCEPTION CLASS APX-EVEX-INVPCID

This exception type is applicable to the EVEX-encoded INVPCID instruction, which is promoted by Intel® APX from legacy space into EVEX map 4.

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present and EVEX.map=4 |
| | | | | X | EVEX prefix is present and EVEX.map=4, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present and EVEX.map=4 and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than V4 is 1.<br>• Any of EVEX.{V4,V3,V2,V1,V0} is 0.<br>• ModRM.Mod=3 |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If the APX_F or any instruction-specific CPUID feature flag is 0. |
| Stack, #SS(0) | | | | X | If a memory address referencing the SS segment is in a non-canonical form. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| | | | | X | If CPL$\neq$0. |
| | | | | X | If an invalid INVPCID_TYPE is specified in the register operand. |
| | | | | X | If INVPCID_DESC[63:12]$\neq$0. |
| | | | | X | If CR4.PCIDE=0, INVPCID_TYPE $\in$ {0,1}, and INVPCID_DESC[11:0]$\neq$0. |
| | | | | X | If INVPCID_TYPE=0 and INVPCID_DESC[127:64] is not a canonical linear address. |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs. |

Table 4.16: Type APX-EVEX-INVPCID Class Exception Conditions

### 4.2.14 EXCEPTION CLASS APX-EVEX-INVVPID

This exception type is applicable to the EVEX-encoded INVVPID instruction, which is promoted by Intel® APX from legacy space into EVEX map 4.

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present and EVEX.map=4 |
| | | | | X | EVEX prefix is present and EVEX.map=4, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present and EVEX.map=4 and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than V4 is 1.<br>• Any of EVEX.{V4,V3,V2,V1,V0} is 0.<br>• ModRM.Mod=3.<br>• Not in VMX operation.<br>• The logical processor does not support VPIDs.<br>• The logical processor supports VPIDs but does not support the INVVPID instruction. |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If the APX_F or any instruction-specific CPUID feature flag is 0. |
| Stack, #SS(0) | | | | X | If a memory address referencing the SS segment is in a non-canonical form. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| | | | | X | If CPL≠0. |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs. |

Table 4.17: Type APX-EVEX-INVVPID Class Exception Conditions

### 4.2.15 EXCEPTION CLASS APX-EVEX-KEYLOCKER

This exception type is applicable to EVEX-encoded Key Locker instructions which are promoted by Intel® APX from legacy space into EVEX map 4 and have the following mnemonics:

ENCODEKEY128, ENCODEKEY256,
AESDEC128KL, AESDEC256KL, AESENC128KL, AESENC256KL,
AESDECWIDE128KL, AESDECWIDE256KL, AESENCWIDE128KL, AESENCWIDE256KL

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present and EVEX.map=4 |
| | | | | X | EVEX prefix is present and EVEX.map=4, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present and EVEX.map=4 and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than V4 is 1.<br>• Any of EVEX.{V4,V3,V2,V1,V0} is 0.<br>• CR0.EM=1 or CR4.OSFXSR=0 (because the original legacy instruction has SSE dependency) or CR4.KL=0.<br>• EVEX.R4=0 for AESDEC128KL, AESDEC256KL, AESENC128KL, or AESENC256KL. |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If the APX_F or any instruction-specific CPUID feature flag is 0. |
| Device Not Available, #NM | | | | X | If CR0.TS=1 |
| Stack, #SS(0) | | | | X | If a memory address referencing the SS segment is in a non-canonical form. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs. |

Table 4.18: Type APX-EVEX-KEYLOCKER Class Exception Conditions

### 4.2.16 EXCEPTION CLASS APX-EVEX-KMOV

This exception type is applicable to EVEX-encoded KMOV* instructions, which are promoted by Intel® APX from VEX space into EVEX space with the same map id (map 1) and opcodes.

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present with a KMOV*'s map id and opcode |
| | | | | X | EVEX prefix is present with a KMOV*'s map id and opcode, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present with a KMOV*'s map id and opcode and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than {V4,L} is 1.<br>• EVEX.L=1 or any of EVEX.{V4,V3,V2,V1,V0} is 0.<br>• Any #UD condition in SDM, vol.2, tables 2-37 and 2-39. |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If the APX_F or any instruction-specific CPUID feature flag is 0. |
| Device Not Available, #NM | | | | X | If CR0.TS=1 |
| Stack, #SS(0) | | | | X | If a memory address referencing the SS segment is in a non-canonical form. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs. |
| Alignment Check #AC(0) | | | | X | If alignment checking is enabled and an unaligned memory access is made while CPL=3 |

Table 4.19: Type APX-EVEX-KMOV Class Exception Conditions

### 4.2.17 EXCEPTION CLASS APX-EVEX-PP2

This exception type is applicable to PUSH2 and POP2 instructions, which are new EVEX map 4 instructions introduced by APX.

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present and EVEX.map=4 |
| | | | | X | EVEX prefix is present and EVEX.map=4, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present and EVEX.map=4 and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than {V4,ND} is 1.<br>• EVEX.ND=0.<br>• ModRM.Mod≠3.<br>• The B register id is 4 (RSP).<br>• The V register id is 4 (RSP).<br>• (For POP2 only) The B and V register ids are the same. |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If CPUID feature flag APX_F is 0. |
| Stack, #SS(0) | | | | X | If a memory address referencing the SS segment is in a non-canonical form. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| | | | | X | If the data being pushed or popped are not 16-byte aligned on the stack. |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs. |

Table 4.20: Type APX-EVEX-PP2 Class Exception Conditions

### 4.2.18 EXCEPTION CLASS APX-EVEX-SHA

This exception type is applicable to EVEX-encoded SHA* instructions which are promoted by Intel® APX from legacy space into EVEX map 4 and have the following mnemonics:

SHA1MSG1, SHA1MSG2, SHA1NEXTE, SHA1RNDS4,
SHA256MSG1, SHA256MSG2, SHA256RNDS2

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | | | EVEX prefix is present. |
| | | | X | | EVEX prefix is present and EVEX.map=4 |
| | | | | X | EVEX prefix is present and EVEX.map=4, but XCR0.APX=0 |
| | | | | X | If EVEX prefix is present and EVEX.map=4 and XCR0.APX=1, but any of the following conditions applies:<br>• In EVEX payload byte 3, any bit other than V4 is 1.<br>• Any of EVEX.{V4,V3,V2,V1,V0} is 0.<br>• CR0.EM=1 or CR4.OSFXSR=0 (because the original legacy instruction has SSE dependency).<br>• EVEX.R4=0 or (EVEX.B4=1 and ModRM.Mod=3) |
| | | | | X | If preceded by any LOCK, 66, F2, F3, or REX prefix |
| | | | | X | If the APX_F or any instruction-specific CPUID feature flag is 0. |
| Device Not Available, #NM | | | | X | If CR0.TS=1 |
| Stack, #SS(0) | | | | X | If a memory address referencing the SS segment is in a non-canonical form. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |
| | | | | X | If the memory address is not 16-byte aligned. |
| Page Fault #PF(faultcode) | | | | X | If a page fault occurs. |

Table 4.21: Type APX-EVEX-SHA Class Exception Conditions

### 4.2.19 EXCEPTION CLASS APX-LEGACY-JMPABS

This exception type is applicable to the JMPABS instruction.

| Exception | Real | Virtual 8086 | Protected & Compatibility | 64-bit | Cause of exception |
|---|---|---|---|---|---|
| Invalid Opcode #UD | X | X | X | | Illegal outside of 64-bit mode |
| | | | | X | If XCR0.APX=0 |
| | | | | X | If preceded by any LOCK, 66, 67, F2, F3, or REX prefix |
| | | | | X | If CPUID feature flag APX_F is 0. |
| General Protection #GP(0) | | | | X | If the memory address is in a non-canonical form. |

Table 4.22: Type APX-LEGACY-JMPABS Class Exception Conditions

# Chapter 5

# INSTRUCTION TABLE

| CPUID: APX_F | OPERANDS | ENCSPACE |
|---|---|---|
| ADC | r8, r8, r8/m8 | EVEX |
| ADC | r8, r8/m8 | EVEX |
| ADC | r8, r8/m8, imm8 | EVEX |
| ADC | r8, r8/m8, r8 | EVEX |
| ADC | r8/m8, imm8 | EVEX |
| ADC | r8/m8, r8 | EVEX |
| ADC | rv, rv, rv/mv | EVEX |
| ADC | rv, rv/mv | EVEX |
| ADC | rv, rv/mv, imm16/imm32 | EVEX |
| ADC | rv, rv/mv, imm8 | EVEX |
| ADC | rv, rv/mv, rv | EVEX |
| ADC | rv/mv, imm16/imm32 | EVEX |
| ADC | rv/mv, imm8 | EVEX |
| ADC | rv/mv, rv | EVEX |
| ADCX | r32, r32, r32/m32 | EVEX |
| ADCX | r32, r32/m32 | EVEX |
| ADCX | r64, r64, r64/m64 | EVEX |
| ADCX | r64, r64/m64 | EVEX |
| ADD | r8, r8, r8/m8 | EVEX |
| ADD | r8, r8/m8 | EVEX |
| ADD | r8, r8/m8, imm8 | EVEX |
| ADD | r8, r8/m8, r8 | EVEX |
| ADD | r8/m8, imm8 | EVEX |
| ADD | r8/m8, r8 | EVEX |
| ADD | rv, rv, rv/mv | EVEX |
| ADD | rv, rv/mv | EVEX |
| ADD | rv, rv/mv, imm16/imm32 | EVEX |
| ADD | rv, rv/mv, imm8 | EVEX |
| ADD | rv, rv/mv, rv | EVEX |
| ADD | rv/mv, imm16/imm32 | EVEX |
| ADD | rv/mv, imm8 | EVEX |
| ADD | rv/mv, rv | EVEX |
| ADOX | r32, r32, r32/m32 | EVEX |
| ADOX | r32, r32/m32 | EVEX |
| ADOX | r64, r64, r64/m64 | EVEX |
| ADOX | r64, r64/m64 | EVEX |
| AESDEC128KL | xmm1, m384 | EVEX |
| AESDEC256KL | xmm1, m512 | EVEX |
| AESDECWIDE128KL | m384, <xmm0:rw>, <xmm1:rw>, <xmm2:rw>, <xmm3:rw>, <xmm4:rw>, <xmm5:rw>, <xmm6:rw>, <xmm7:rw> | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| AESDECWIDE256KL | m512, <xmm0:rw>, <xmm1:rw>, <xmm2:rw>, <xmm3:rw>, <xmm4:rw>, <xmm5:rw>, <xmm6:rw>, <xmm7:rw> | EVEX |
| AESENC128KL | xmm1, m384 | EVEX |
| AESENC256KL | xmm1, m512 | EVEX |
| AESENCWIDE128KL | m384, <xmm0:rw>, <xmm1:rw>, <xmm2:rw>, <xmm3:rw>, <xmm4:rw>, <xmm5:rw>, <xmm6:rw>, <xmm7:rw> | EVEX |
| AESENCWIDE256KL | m512, <xmm0:rw>, <xmm1:rw>, <xmm2:rw>, <xmm3:rw>, <xmm4:rw>, <xmm5:rw>, <xmm6:rw>, <xmm7:rw> | EVEX |
| AND | r8, r8, r8/m8 | EVEX |
| AND | r8, r8/m8 | EVEX |
| AND | r8, r8/m8, imm8 | EVEX |
| AND | r8, r8/m8, r8 | EVEX |
| AND | r8/m8, imm8 | EVEX |
| AND | r8/m8, r8 | EVEX |
| AND | rv, rv, rv/mv | EVEX |
| AND | rv, rv/mv | EVEX |
| AND | rv, rv/mv, imm16/imm32 | EVEX |
| AND | rv, rv/mv, imm8 | EVEX |
| AND | rv, rv/mv, rv | EVEX |
| AND | rv/mv, imm16/imm32 | EVEX |
| AND | rv/mv, imm8 | EVEX |
| AND | rv/mv, rv | EVEX |
| ANDN | r32, r32, m32/r32 | EVEX |
| ANDN | r64, r64, m64/r64 | EVEX |
| BEXTR | r32, m32/r32, r32 | EVEX |
| BEXTR | r64, m64/r64, r64 | EVEX |
| BLSI | r32, m32/r32 | EVEX |
| BLSI | r64, m64/r64 | EVEX |
| BLSMSK | r32, m32/r32 | EVEX |
| BLSMSK | r64, m64/r64 | EVEX |
| BLSR | r32, m32/r32 | EVEX |
| BLSR | r64, m64/r64 | EVEX |
| BZHI | r32, m32/r32, r32 | EVEX |
| BZHI | r64, m64/r64, r64 | EVEX |
| CCMPB | r8, r8/m8, dfv | EVEX |
| CCMPB | r8/m8, imm8, dfv | EVEX |
| CCMPB | r8/m8, r8, dfv | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| CCMPB | rv, rv/mv, dfv | EVEX |
| CCMPB | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPB | rv/mv, imm8, dfv | EVEX |
| CCMPB | rv/mv, rv, dfv | EVEX |
| CCMPBE | r8, r8/m8, dfv | EVEX |
| CCMPBE | r8/m8, imm8, dfv | EVEX |
| CCMPBE | r8/m8, r8, dfv | EVEX |
| CCMPBE | rv, rv/mv, dfv | EVEX |
| CCMPBE | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPBE | rv/mv, imm8, dfv | EVEX |
| CCMPBE | rv/mv, rv, dfv | EVEX |
| CCMPF | r8, r8/m8, dfv | EVEX |
| CCMPF | r8/m8, imm8, dfv | EVEX |
| CCMPF | r8/m8, r8, dfv | EVEX |
| CCMPF | rv, rv/mv, dfv | EVEX |
| CCMPF | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPF | rv/mv, imm8, dfv | EVEX |
| CCMPF | rv/mv, rv, dfv | EVEX |
| CCMPL | r8, r8/m8, dfv | EVEX |
| CCMPL | r8/m8, imm8, dfv | EVEX |
| CCMPL | r8/m8, r8, dfv | EVEX |
| CCMPL | rv, rv/mv, dfv | EVEX |
| CCMPL | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPL | rv/mv, imm8, dfv | EVEX |
| CCMPL | rv/mv, rv, dfv | EVEX |
| CCMPLE | r8, r8/m8, dfv | EVEX |
| CCMPLE | r8/m8, imm8, dfv | EVEX |
| CCMPLE | r8/m8, r8, dfv | EVEX |
| CCMPLE | rv, rv/mv, dfv | EVEX |
| CCMPLE | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPLE | rv/mv, imm8, dfv | EVEX |
| CCMPLE | rv/mv, rv, dfv | EVEX |
| CCMPNB | r8, r8/m8, dfv | EVEX |
| CCMPNB | r8/m8, imm8, dfv | EVEX |
| CCMPNB | r8/m8, r8, dfv | EVEX |
| CCMPNB | rv, rv/mv, dfv | EVEX |
| CCMPNB | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPNB | rv/mv, imm8, dfv | EVEX |
| CCMPNB | rv/mv, rv, dfv | EVEX |
| CCMPNBE | r8, r8/m8, dfv | EVEX |
| CCMPNBE | r8/m8, imm8, dfv | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| CCMPNBE | r8/m8, r8, dfv | EVEX |
| CCMPNBE | rv, rv/mv, dfv | EVEX |
| CCMPNBE | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPNBE | rv/mv, imm8, dfv | EVEX |
| CCMPNBE | rv/mv, rv, dfv | EVEX |
| CCMPNL | r8, r8/m8, dfv | EVEX |
| CCMPNL | r8/m8, imm8, dfv | EVEX |
| CCMPNL | r8/m8, r8, dfv | EVEX |
| CCMPNL | rv, rv/mv, dfv | EVEX |
| CCMPNL | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPNL | rv/mv, imm8, dfv | EVEX |
| CCMPNL | rv/mv, rv, dfv | EVEX |
| CCMPNLE | r8, r8/m8, dfv | EVEX |
| CCMPNLE | r8/m8, imm8, dfv | EVEX |
| CCMPNLE | r8/m8, r8, dfv | EVEX |
| CCMPNLE | rv, rv/mv, dfv | EVEX |
| CCMPNLE | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPNLE | rv/mv, imm8, dfv | EVEX |
| CCMPNLE | rv/mv, rv, dfv | EVEX |
| CCMPNO | r8, r8/m8, dfv | EVEX |
| CCMPNO | r8/m8, imm8, dfv | EVEX |
| CCMPNO | r8/m8, r8, dfv | EVEX |
| CCMPNO | rv, rv/mv, dfv | EVEX |
| CCMPNO | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPNO | rv/mv, imm8, dfv | EVEX |
| CCMPNO | rv/mv, rv, dfv | EVEX |
| CCMPNS | r8, r8/m8, dfv | EVEX |
| CCMPNS | r8/m8, imm8, dfv | EVEX |
| CCMPNS | r8/m8, r8, dfv | EVEX |
| CCMPNS | rv, rv/mv, dfv | EVEX |
| CCMPNS | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPNS | rv/mv, imm8, dfv | EVEX |
| CCMPNS | rv/mv, rv, dfv | EVEX |
| CCMPNZ | r8, r8/m8, dfv | EVEX |
| CCMPNZ | r8/m8, imm8, dfv | EVEX |
| CCMPNZ | r8/m8, r8, dfv | EVEX |
| CCMPNZ | rv, rv/mv, dfv | EVEX |
| CCMPNZ | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPNZ | rv/mv, imm8, dfv | EVEX |
| CCMPNZ | rv/mv, rv, dfv | EVEX |
| CCMPO | r8, r8/m8, dfv | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| CCMPO | r8/m8, imm8, dfv | EVEX |
| CCMPO | r8/m8, r8, dfv | EVEX |
| CCMPO | rv, rv/mv, dfv | EVEX |
| CCMPO | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPO | rv/mv, imm8, dfv | EVEX |
| CCMPO | rv/mv, rv, dfv | EVEX |
| CCMPS | r8, r8/m8, dfv | EVEX |
| CCMPS | r8/m8, imm8, dfv | EVEX |
| CCMPS | r8/m8, r8, dfv | EVEX |
| CCMPS | rv, rv/mv, dfv | EVEX |
| CCMPS | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPS | rv/mv, imm8, dfv | EVEX |
| CCMPS | rv/mv, rv, dfv | EVEX |
| CCMPT | r8, r8/m8, dfv | EVEX |
| CCMPT | r8/m8, imm8, dfv | EVEX |
| CCMPT | r8/m8, r8, dfv | EVEX |
| CCMPT | rv, rv/mv, dfv | EVEX |
| CCMPT | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPT | rv/mv, imm8, dfv | EVEX |
| CCMPT | rv/mv, rv, dfv | EVEX |
| CCMPZ | r8, r8/m8, dfv | EVEX |
| CCMPZ | r8/m8, imm8, dfv | EVEX |
| CCMPZ | r8/m8, r8, dfv | EVEX |
| CCMPZ | rv, rv/mv, dfv | EVEX |
| CCMPZ | rv/mv, imm16/imm32, dfv | EVEX |
| CCMPZ | rv/mv, imm8, dfv | EVEX |
| CCMPZ | rv/mv, rv, dfv | EVEX |
| CFCMOVB | mv, rv | EVEX |
| CFCMOVB | rv, rv | EVEX |
| CFCMOVB | rv, rv, rv/mv | EVEX |
| CFCMOVB | rv, rv/mv | EVEX |
| CFCMOVBE | mv, rv | EVEX |
| CFCMOVBE | rv, rv | EVEX |
| CFCMOVBE | rv, rv, rv/mv | EVEX |
| CFCMOVBE | rv, rv/mv | EVEX |
| CFCMOVL | mv, rv | EVEX |
| CFCMOVL | rv, rv | EVEX |
| CFCMOVL | rv, rv, rv/mv | EVEX |
| CFCMOVL | rv, rv/mv | EVEX |
| CFCMOVLE | mv, rv | EVEX |
| CFCMOVLE | rv, rv | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| CFCMOVLE | rv, rv, rv/mv | EVEX |
| CFCMOVLE | rv, rv/mv | EVEX |
| CFCMOVNB | mv, rv | EVEX |
| CFCMOVNB | rv, rv | EVEX |
| CFCMOVNB | rv, rv, rv/mv | EVEX |
| CFCMOVNB | rv, rv/mv | EVEX |
| CFCMOVNBE | mv, rv | EVEX |
| CFCMOVNBE | rv, rv | EVEX |
| CFCMOVNBE | rv, rv, rv/mv | EVEX |
| CFCMOVNBE | rv, rv/mv | EVEX |
| CFCMOVNL | mv, rv | EVEX |
| CFCMOVNL | rv, rv | EVEX |
| CFCMOVNL | rv, rv, rv/mv | EVEX |
| CFCMOVNL | rv, rv/mv | EVEX |
| CFCMOVNLE | mv, rv | EVEX |
| CFCMOVNLE | rv, rv | EVEX |
| CFCMOVNLE | rv, rv, rv/mv | EVEX |
| CFCMOVNLE | rv, rv/mv | EVEX |
| CFCMOVNO | mv, rv | EVEX |
| CFCMOVNO | rv, rv | EVEX |
| CFCMOVNO | rv, rv, rv/mv | EVEX |
| CFCMOVNO | rv, rv/mv | EVEX |
| CFCMOVNP | mv, rv | EVEX |
| CFCMOVNP | rv, rv | EVEX |
| CFCMOVNP | rv, rv, rv/mv | EVEX |
| CFCMOVNP | rv, rv/mv | EVEX |
| CFCMOVNS | mv, rv | EVEX |
| CFCMOVNS | rv, rv | EVEX |
| CFCMOVNS | rv, rv, rv/mv | EVEX |
| CFCMOVNS | rv, rv/mv | EVEX |
| CFCMOVNZ | mv, rv | EVEX |
| CFCMOVNZ | rv, rv | EVEX |
| CFCMOVNZ | rv, rv, rv/mv | EVEX |
| CFCMOVNZ | rv, rv/mv | EVEX |
| CFCMOVO | mv, rv | EVEX |
| CFCMOVO | rv, rv | EVEX |
| CFCMOVO | rv, rv, rv/mv | EVEX |
| CFCMOVO | rv, rv/mv | EVEX |
| CFCMOVP | mv, rv | EVEX |
| CFCMOVP | rv, rv | EVEX |
| CFCMOVP | rv, rv, rv/mv | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| CFCMOVP | rv, rv/mv | EVEX |
| CFCMOVS | mv, rv | EVEX |
| CFCMOVS | rv, rv | EVEX |
| CFCMOVS | rv, rv, rv/mv | EVEX |
| CFCMOVS | rv, rv/mv | EVEX |
| CFCMOVZ | mv, rv | EVEX |
| CFCMOVZ | rv, rv | EVEX |
| CFCMOVZ | rv, rv, rv/mv | EVEX |
| CFCMOVZ | rv, rv/mv | EVEX |
| CMOVB | rv, rv, rv/mv | EVEX |
| CMOVBE | rv, rv, rv/mv | EVEX |
| CMOVL | rv, rv, rv/mv | EVEX |
| CMOVLE | rv, rv, rv/mv | EVEX |
| CMOVNB | rv, rv, rv/mv | EVEX |
| CMOVNBE | rv, rv, rv/mv | EVEX |
| CMOVNL | rv, rv, rv/mv | EVEX |
| CMOVNLE | rv, rv, rv/mv | EVEX |
| CMOVNO | rv, rv, rv/mv | EVEX |
| CMOVNP | rv, rv, rv/mv | EVEX |
| CMOVNS | rv, rv, rv/mv | EVEX |
| CMOVNZ | rv, rv, rv/mv | EVEX |
| CMOVO | rv, rv, rv/mv | EVEX |
| CMOVP | rv, rv, rv/mv | EVEX |
| CMOVS | rv, rv, rv/mv | EVEX |
| CMOVZ | rv, rv, rv/mv | EVEX |
| CMPBEXADD | m32, r32, r32 | EVEX |
| CMPBEXADD | m64, r64, r64 | EVEX |
| CMPBXADD | m32, r32, r32 | EVEX |
| CMPBXADD | m64, r64, r64 | EVEX |
| CMPLEXADD | m32, r32, r32 | EVEX |
| CMPLEXADD | m64, r64, r64 | EVEX |
| CMPLXADD | m32, r32, r32 | EVEX |
| CMPLXADD | m64, r64, r64 | EVEX |
| CMPNBEXADD | m32, r32, r32 | EVEX |
| CMPNBEXADD | m64, r64, r64 | EVEX |
| CMPNBXADD | m32, r32, r32 | EVEX |
| CMPNBXADD | m64, r64, r64 | EVEX |
| CMPNLEXADD | m32, r32, r32 | EVEX |
| CMPNLEXADD | m64, r64, r64 | EVEX |
| CMPNLXADD | m32, r32, r32 | EVEX |
| CMPNLXADD | m64, r64, r64 | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| CMPNOXADD | m32, r32, r32 | EVEX |
| CMPNOXADD | m64, r64, r64 | EVEX |
| CMPNPXADD | m32, r32, r32 | EVEX |
| CMPNPXADD | m64, r64, r64 | EVEX |
| CMPNSXADD | m32, r32, r32 | EVEX |
| CMPNSXADD | m64, r64, r64 | EVEX |
| CMPNZXADD | m32, r32, r32 | EVEX |
| CMPNZXADD | m64, r64, r64 | EVEX |
| CMPOXADD | m32, r32, r32 | EVEX |
| CMPOXADD | m64, r64, r64 | EVEX |
| CMPPXADD | m32, r32, r32 | EVEX |
| CMPPXADD | m64, r64, r64 | EVEX |
| CMPSXADD | m32, r32, r32 | EVEX |
| CMPSXADD | m64, r64, r64 | EVEX |
| CMPZXADD | m32, r32, r32 | EVEX |
| CMPZXADD | m64, r64, r64 | EVEX |
| CRC32 | ry, r8/m8 | EVEX |
| CRC32 | ry, rv/mv | EVEX |
| CTESTB | r8/m8, imm8, dfv | EVEX |
| CTESTB | r8/m8, r8, dfv | EVEX |
| CTESTB | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTB | rv/mv, rv, dfv | EVEX |
| CTESTBE | r8/m8, imm8, dfv | EVEX |
| CTESTBE | r8/m8, r8, dfv | EVEX |
| CTESTBE | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTBE | rv/mv, rv, dfv | EVEX |
| CTESTF | r8/m8, imm8, dfv | EVEX |
| CTESTF | r8/m8, r8, dfv | EVEX |
| CTESTF | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTF | rv/mv, rv, dfv | EVEX |
| CTESTL | r8/m8, imm8, dfv | EVEX |
| CTESTL | r8/m8, r8, dfv | EVEX |
| CTESTL | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTL | rv/mv, rv, dfv | EVEX |
| CTESTLE | r8/m8, imm8, dfv | EVEX |
| CTESTLE | r8/m8, r8, dfv | EVEX |
| CTESTLE | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTLE | rv/mv, rv, dfv | EVEX |
| CTESTNB | r8/m8, imm8, dfv | EVEX |
| CTESTNB | r8/m8, r8, dfv | EVEX |
| CTESTNB | rv/mv, imm16/imm32, dfv | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| CTESTNB | rv/mv, rv, dfv | EVEX |
| CTESTNBE | r8/m8, imm8, dfv | EVEX |
| CTESTNBE | r8/m8, r8, dfv | EVEX |
| CTESTNBE | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTNBE | rv/mv, rv, dfv | EVEX |
| CTESTNL | r8/m8, imm8, dfv | EVEX |
| CTESTNL | r8/m8, r8, dfv | EVEX |
| CTESTNL | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTNL | rv/mv, rv, dfv | EVEX |
| CTESTNLE | r8/m8, imm8, dfv | EVEX |
| CTESTNLE | r8/m8, r8, dfv | EVEX |
| CTESTNLE | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTNLE | rv/mv, rv, dfv | EVEX |
| CTESTNO | r8/m8, imm8, dfv | EVEX |
| CTESTNO | r8/m8, r8, dfv | EVEX |
| CTESTNO | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTNO | rv/mv, rv, dfv | EVEX |
| CTESTNS | r8/m8, imm8, dfv | EVEX |
| CTESTNS | r8/m8, r8, dfv | EVEX |
| CTESTNS | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTNS | rv/mv, rv, dfv | EVEX |
| CTESTNZ | r8/m8, imm8, dfv | EVEX |
| CTESTNZ | r8/m8, r8, dfv | EVEX |
| CTESTNZ | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTNZ | rv/mv, rv, dfv | EVEX |
| CTESTO | r8/m8, imm8, dfv | EVEX |
| CTESTO | r8/m8, r8, dfv | EVEX |
| CTESTO | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTO | rv/mv, rv, dfv | EVEX |
| CTESTS | r8/m8, imm8, dfv | EVEX |
| CTESTS | r8/m8, r8, dfv | EVEX |
| CTESTS | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTS | rv/mv, rv, dfv | EVEX |
| CTESTT | r8/m8, imm8, dfv | EVEX |
| CTESTT | r8/m8, r8, dfv | EVEX |
| CTESTT | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTT | rv/mv, rv, dfv | EVEX |
| CTESTZ | r8/m8, imm8, dfv | EVEX |
| CTESTZ | r8/m8, r8, dfv | EVEX |
| CTESTZ | rv/mv, imm16/imm32, dfv | EVEX |
| CTESTZ | rv/mv, rv, dfv | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| DEC | r8, r8/m8 | EVEX |
| DEC | r8/m8 | EVEX |
| DEC | rv, rv/mv | EVEX |
| DEC | rv/mv | EVEX |
| DIV | r8/m8, <ax:rw:supp> | EVEX |
| DIV | rv/mv, <orax:rw:supp>, <ordx:rw:supp> | EVEX |
| ENCODEKEY128 | r32, r32, <xmm0:rw>, <xmm1:w>, <xmm2:w>, <xmm4:w>, <xmm5:w>, <xmm6:w> | EVEX |
| ENCODEKEY256 | r32, r32, <xmm0:rw>, <xmm1:rw>, <xmm2:w>, <xmm3:w>, <xmm4:w>, <xmm5:w>, <xmm6:w> | EVEX |
| ENQCMD | ra, m512 | EVEX |
| ENQCMDS | ra, m512 | EVEX |
| IDIV | r8/m8, <ax:rw:supp> | EVEX |
| IDIV | rv/mv, <orax:rw:supp>, <ordx:rw:supp> | EVEX |
| IMUL | r8/m8, <al:r:supp>, <ax:w:supp> | EVEX |
| IMUL | rv, rv, rv/mv | EVEX |
| IMUL | rv, rv/mv | EVEX |
| IMUL | rv, rv/mv, imm16/imm32 | EVEX |
| IMUL | rv, rv/mv, imm8 | EVEX |
| IMUL | rv/mv, <orax:rw:supp>, <ordx:w:supp> | EVEX |
| INC | r8, r8/m8 | EVEX |
| INC | r8/m8 | EVEX |
| INC | rv, rv/mv | EVEX |
| INC | rv/mv | EVEX |
| INVEPT | r64, m128 | EVEX |
| INVPCID | r64, m128 | EVEX |
| INVVPID | r64, m128 | EVEX |
| JMPABS | target64 | LEGACY |
| KMOVB | k1, k2/m8 | EVEX |
| KMOVB | k1, r32 | EVEX |
| KMOVB | m8, k1 | EVEX |
| KMOVB | r32, k1 | EVEX |
| KMOVD | k1, k2/m32 | EVEX |
| KMOVD | k1, r32 | EVEX |
| KMOVD | m32, k1 | EVEX |
| KMOVD | r32, k1 | EVEX |
| KMOVQ | k1, k2/m64 | EVEX |
| KMOVQ | k1, r64 | EVEX |
| KMOVQ | m64, k1 | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| KMOVQ | r64, k1 | EVEX |
| KMOVW | k1, k2/m16 | EVEX |
| KMOVW | k1, r32 | EVEX |
| KMOVW | m16, k1 | EVEX |
| KMOVW | r32, k1 | EVEX |
| LZCNT | rv, rv/mv | EVEX |
| MOVBE | rv, rv/mv | EVEX |
| MOVBE | rv/mv, rv | EVEX |
| MOVDIR64B | ra, m512 | EVEX |
| MOVDIRI | my, ry | EVEX |
| MUL | r8/m8, <al:r:supp>, <ax:w:supp> | EVEX |
| MUL | rv/mv, <orax:rw:supp>, <ordx:w:supp> | EVEX |
| MULX | r32, r32, m32/r32, <edx:r:supp> | EVEX |
| MULX | r64, r64, m64/r64, <rdx:r:supp> | EVEX |
| NEG | r8, r8/m8 | EVEX |
| NEG | r8/m8 | EVEX |
| NEG | rv, rv/mv | EVEX |
| NEG | rv/mv | EVEX |
| NOT | r8, r8/m8 | EVEX |
| NOT | r8/m8 | EVEX |
| NOT | rv, rv/mv | EVEX |
| NOT | rv/mv | EVEX |
| OR | r8, r8, r8/m8 | EVEX |
| OR | r8, r8/m8 | EVEX |
| OR | r8, r8/m8, imm8 | EVEX |
| OR | r8, r8/m8, r8 | EVEX |
| OR | r8/m8, imm8 | EVEX |
| OR | r8/m8, r8 | EVEX |
| OR | rv, rv, rv/mv | EVEX |
| OR | rv, rv/mv | EVEX |
| OR | rv, rv/mv, imm16/imm32 | EVEX |
| OR | rv, rv/mv, imm8 | EVEX |
| OR | rv, rv/mv, rv | EVEX |
| OR | rv/mv, imm16/imm32 | EVEX |
| OR | rv/mv, imm8 | EVEX |
| OR | rv/mv, rv | EVEX |
| PDEP | r32, r32, m32/r32 | EVEX |
| PDEP | r64, r64, m64/r64 | EVEX |
| PEXT | r32, r32, m32/r32 | EVEX |
| PEXT | r64, r64, m64/r64 | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| POP2 | r64, r64, <pop:rw:supp> | EVEX |
| POP2P | r64, r64, <pop:rw:supp> | EVEX |
| POPCNT | rv, rv/mv | EVEX |
| PUSH2 | r64, r64, <push:rw:supp> | EVEX |
| PUSH2P | r64, r64, <push:rw:supp> | EVEX |
| RCL | r8, r8/m8, <1:r:impl> | EVEX |
| RCL | r8, r8/m8, <cl:r:impl> | EVEX |
| RCL | r8, r8/m8, imm8 | EVEX |
| RCL | r8/m8, <1:r:impl> | EVEX |
| RCL | r8/m8, <cl:r:impl> | EVEX |
| RCL | r8/m8, imm8 | EVEX |
| RCL | rv, rv/mv, <1:r:impl> | EVEX |
| RCL | rv, rv/mv, <cl:r:impl> | EVEX |
| RCL | rv, rv/mv, imm8 | EVEX |
| RCL | rv/mv, <1:r:impl> | EVEX |
| RCL | rv/mv, <cl:r:impl> | EVEX |
| RCL | rv/mv, imm8 | EVEX |
| RCR | r8, r8/m8, <1:r:impl> | EVEX |
| RCR | r8, r8/m8, <cl:r:impl> | EVEX |
| RCR | r8, r8/m8, imm8 | EVEX |
| RCR | r8/m8, <1:r:impl> | EVEX |
| RCR | r8/m8, <cl:r:impl> | EVEX |
| RCR | r8/m8, imm8 | EVEX |
| RCR | rv, rv/mv, <1:r:impl> | EVEX |
| RCR | rv, rv/mv, <cl:r:impl> | EVEX |
| RCR | rv, rv/mv, imm8 | EVEX |
| RCR | rv/mv, <1:r:impl> | EVEX |
| RCR | rv/mv, <cl:r:impl> | EVEX |
| RCR | rv/mv, imm8 | EVEX |
| ROL | r8, r8/m8, <1:r:impl> | EVEX |
| ROL | r8, r8/m8, <cl:r:impl> | EVEX |
| ROL | r8, r8/m8, imm8 | EVEX |
| ROL | r8/m8, <1:r:impl> | EVEX |
| ROL | r8/m8, <cl:r:impl> | EVEX |
| ROL | r8/m8, imm8 | EVEX |
| ROL | rv, rv/mv, <1:r:impl> | EVEX |
| ROL | rv, rv/mv, <cl:r:impl> | EVEX |
| ROL | rv, rv/mv, imm8 | EVEX |
| ROL | rv/mv, <1:r:impl> | EVEX |
| ROL | rv/mv, <cl:r:impl> | EVEX |
| ROL | rv/mv, imm8 | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| ROR | r8, r8/m8, <1:r:impl> | EVEX |
| ROR | r8, r8/m8, <cl:r:impl> | EVEX |
| ROR | r8, r8/m8, imm8 | EVEX |
| ROR | r8/m8, <1:r:impl> | EVEX |
| ROR | r8/m8, <cl:r:impl> | EVEX |
| ROR | r8/m8, imm8 | EVEX |
| ROR | rv, rv/mv, <1:r:impl> | EVEX |
| ROR | rv, rv/mv, <cl:r:impl> | EVEX |
| ROR | rv, rv/mv, imm8 | EVEX |
| ROR | rv/mv, <1:r:impl> | EVEX |
| ROR | rv/mv, <cl:r:impl> | EVEX |
| ROR | rv/mv, imm8 | EVEX |
| RORX | r32, m32/r32, imm8 | EVEX |
| RORX | r64, m64/r64, imm8 | EVEX |
| SAR | r8, r8/m8, <1:r:impl> | EVEX |
| SAR | r8, r8/m8, <cl:r:impl> | EVEX |
| SAR | r8, r8/m8, imm8 | EVEX |
| SAR | r8/m8, <1:r:impl> | EVEX |
| SAR | r8/m8, <cl:r:impl> | EVEX |
| SAR | r8/m8, imm8 | EVEX |
| SAR | rv, rv/mv, <1:r:impl> | EVEX |
| SAR | rv, rv/mv, <cl:r:impl> | EVEX |
| SAR | rv, rv/mv, imm8 | EVEX |
| SAR | rv/mv, <1:r:impl> | EVEX |
| SAR | rv/mv, <cl:r:impl> | EVEX |
| SAR | rv/mv, imm8 | EVEX |
| SARX | r32, m32/r32, r32 | EVEX |
| SARX | r64, m64/r64, r64 | EVEX |
| SBB | r8, r8, r8/m8 | EVEX |
| SBB | r8, r8/m8 | EVEX |
| SBB | r8, r8/m8, imm8 | EVEX |
| SBB | r8, r8/m8, r8 | EVEX |
| SBB | r8/m8, imm8 | EVEX |
| SBB | r8/m8, r8 | EVEX |
| SBB | rv, rv, rv/mv | EVEX |
| SBB | rv, rv/mv | EVEX |
| SBB | rv, rv/mv, imm16/imm32 | EVEX |
| SBB | rv, rv/mv, imm8 | EVEX |
| SBB | rv, rv/mv, rv | EVEX |
| SBB | rv/mv, imm16/imm32 | EVEX |
| SBB | rv/mv, imm8 | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| SBB | rv/mv, rv | EVEX |
| SETB | r8/m8 | EVEX |
| SETBE | r8/m8 | EVEX |
| SETL | r8/m8 | EVEX |
| SETLE | r8/m8 | EVEX |
| SETNB | r8/m8 | EVEX |
| SETNBE | r8/m8 | EVEX |
| SETNL | r8/m8 | EVEX |
| SETNLE | r8/m8 | EVEX |
| SETNO | r8/m8 | EVEX |
| SETNP | r8/m8 | EVEX |
| SETNS | r8/m8 | EVEX |
| SETNZ | r8/m8 | EVEX |
| SETO | r8/m8 | EVEX |
| SETP | r8/m8 | EVEX |
| SETS | r8/m8 | EVEX |
| SETZ | r8/m8 | EVEX |
| SHA1MSG1 | xmm1, xmm2/m128 | EVEX |
| SHA1MSG2 | xmm1, xmm2/m128 | EVEX |
| SHA1NEXTE | xmm1, xmm2/m128 | EVEX |
| SHA1RNDS4 | xmm1, xmm2/m128, imm8 | EVEX |
| SHA256MSG1 | xmm1, xmm2/m128 | EVEX |
| SHA256MSG2 | xmm1, xmm2/m128 | EVEX |
| SHA256RNDS2 | xmm1, xmm2/m128, <xmm0> | EVEX |
| SHL | r8, r8/m8, <1:r:impl> | EVEX |
| SHL | r8, r8/m8, <cl:r:impl> | EVEX |
| SHL | r8, r8/m8, imm8 | EVEX |
| SHL | r8/m8, <1:r:impl> | EVEX |
| SHL | r8/m8, <cl:r:impl> | EVEX |
| SHL | r8/m8, imm8 | EVEX |
| SHL | rv, rv/mv, <1:r:impl> | EVEX |
| SHL | rv, rv/mv, <cl:r:impl> | EVEX |
| SHL | rv, rv/mv, imm8 | EVEX |
| SHL | rv/mv, <1:r:impl> | EVEX |
| SHL | rv/mv, <cl:r:impl> | EVEX |
| SHL | rv/mv, imm8 | EVEX |
| SHLD | rv, rv/mv, rv, <cl:r:impl> | EVEX |
| SHLD | rv, rv/mv, rv, imm8 | EVEX |
| SHLD | rv/mv, rv, <cl:r:impl> | EVEX |
| SHLD | rv/mv, rv, imm8 | EVEX |
| SHLX | r32, m32/r32, r32 | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| SHLX | r64, m64/r64, r64 | EVEX |
| SHR | r8, r8/m8, <1:r:impl> | EVEX |
| SHR | r8, r8/m8, <cl:r:impl> | EVEX |
| SHR | r8, r8/m8, imm8 | EVEX |
| SHR | r8/m8, <1:r:impl> | EVEX |
| SHR | r8/m8, <cl:r:impl> | EVEX |
| SHR | r8/m8, imm8 | EVEX |
| SHR | rv, rv/mv, <1:r:impl> | EVEX |
| SHR | rv, rv/mv, <cl:r:impl> | EVEX |
| SHR | rv, rv/mv, imm8 | EVEX |
| SHR | rv/mv, <1:r:impl> | EVEX |
| SHR | rv/mv, <cl:r:impl> | EVEX |
| SHR | rv/mv, imm8 | EVEX |
| SHRD | rv, rv/mv, rv, <cl:r:impl> | EVEX |
| SHRD | rv, rv/mv, rv, imm8 | EVEX |
| SHRD | rv/mv, rv, <cl:r:impl> | EVEX |
| SHRD | rv/mv, rv, imm8 | EVEX |
| SHRX | r32, m32/r32, r32 | EVEX |
| SHRX | r64, m64/r64, r64 | EVEX |
| SUB | r8, r8, r8/m8 | EVEX |
| SUB | r8, r8/m8 | EVEX |
| SUB | r8, r8/m8, imm8 | EVEX |
| SUB | r8, r8/m8, r8 | EVEX |
| SUB | r8/m8, imm8 | EVEX |
| SUB | r8/m8, r8 | EVEX |
| SUB | rv, rv, rv/mv | EVEX |
| SUB | rv, rv/mv | EVEX |
| SUB | rv, rv/mv, imm16/imm32 | EVEX |
| SUB | rv, rv/mv, imm8 | EVEX |
| SUB | rv, rv/mv, rv | EVEX |
| SUB | rv/mv, imm16/imm32 | EVEX |
| SUB | rv/mv, imm8 | EVEX |
| SUB | rv/mv, rv | EVEX |
| TZCNT | rv, rv/mv | EVEX |
| WRSSD | m32, r32 | EVEX |
| WRSSQ | m64, r64 | EVEX |
| WRUSSD | m32, r32 | EVEX |
| WRUSSQ | m64, r64 | EVEX |
| XOR | r8, r8, r8/m8 | EVEX |
| XOR | r8, r8/m8 | EVEX |
| XOR | r8, r8/m8, imm8 | EVEX |

*Table continued on next page...*

| CPUID: APX_F | OPERANDS | ENCSPACE *(contd.)* |
|---|---|---|
| XOR | r8, r8/m8, r8 | EVEX |
| XOR | r8/m8, imm8 | EVEX |
| XOR | r8/m8, r8 | EVEX |
| XOR | rv, rv, rv/mv | EVEX |
| XOR | rv, rv/mv | EVEX |
| XOR | rv, rv/mv, imm16/imm32 | EVEX |
| XOR | rv, rv/mv, imm8 | EVEX |
| XOR | rv, rv/mv, rv | EVEX |
| XOR | rv/mv, imm16/imm32 | EVEX |
| XOR | rv/mv, imm8 | EVEX |
| XOR | rv/mv, rv | EVEX |
| **CPUID: APX-F-AMX** | **OPERANDS** | **ENCSPACE** |
| LDTILECFG | m512 | EVEX |
| STTILECFG | m512 | EVEX |
| TILELOADD | tmm1, sibmem | EVEX |
| TILELOADDT1 | tmm1, sibmem | EVEX |
| TILESTORED | sibmem, tmm1 | EVEX |

# Chapter 6

# INTEL® APX EXTENDED INSTRUCTIONS

## 6.1 ADC

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED 10 /r<br>ADC {NF=0} {ND=0} r8/m8, r8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 10 /r<br>ADC {NF=0} {ND=1} r8, r8/m8, r8 | F | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 11 /r<br>ADC {NF=0} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 11 /r<br>ADC {NF=0} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 11 /r<br>ADC {NF=0} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 11 /r<br>ADC {NF=0} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 12 /r<br>ADC {NF=0} {ND=0} r8, r8/m8 | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 12 /r<br>ADC {NF=0} {ND=1} r8, r8, r8/m8 | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 13 /r<br>ADC {NF=0} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 13 /r<br>ADC {NF=0} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 13 /r<br>ADC {NF=0} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 13 /r<br>ADC {NF=0} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /2 ib<br>ADC {NF=0} {ND=0} r8/m8, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /2 ib<br>ADC {NF=0} {ND=1} r8, r8/m8, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /2 id<br>ADC {NF=0} {ND=0} rv/mv, imm32 | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 81 /2 iw/id<br>ADC {NF=0} {ND=0} rv/mv, imm16/imm32 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /2 id<br>ADC {NF=0} {ND=1} rv, rv/mv, imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /2 iw/id<br>ADC {NF=0} {ND=1} rv, rv/mv, imm16/imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /2 ib<br>ADC {NF=0} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /2 ib<br>ADC {NF=0} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /2 ib<br>ADC {NF=0} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /2 ib<br>ADC {NF=0} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |

### 6.1.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | MODRM.REG(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |
| C | NO-SCALE | MODRM.R/M(rw) | IMM16/IMM32(r) | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM16/IMM32(r) | N/A |
| E | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| F | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | MODRM.REG(r) | N/A |
| G | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |
| H | NO-SCALE | VVVVV(w) | MODRM.REG(r) | MODRM.R/M(r) | N/A |

### 6.1.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.1.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| ADC r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| ADC r8, r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| ADC rv/mv, rv | APX-EVEX-INT | N/A | APX_F |
| ADC rv, rv/mv, rv | APX-EVEX-INT | N/A | APX_F |
| ADC r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| ADC r8, r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| ADC rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| ADC rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| ADC r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| ADC r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| ADC rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| ADC rv, rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| ADC rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| ADC rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |

## 6.2 ADCX

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.W0 66 /r<br>ADCX {NF=0} {ND=0} r32, r32/m32 | A | V/N.E. | APX_F<br>ADX |
| EVEX.LLZ.66.MAP4.W1 66 /r<br>ADCX {NF=0} {ND=0} r64, r64/m64 | A | V/N.E. | APX_F<br>ADX |
| EVEX.LLZ.66.MAP4.W0 66 /r<br>ADCX {NF=0} {ND=1} r32, r32, r32/m32 | B | V/N.E. | APX_F<br>ADX |
| EVEX.LLZ.66.MAP4.W1 66 /r<br>ADCX {NF=0} {ND=1} r64, r64, r64/m64 | B | V/N.E. | APX_F<br>ADX |

### 6.2.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.REG(r) | MODRM.R/M(r) | N/A |

### 6.2.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.2.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| ADCX r32, r32/m32 | APX-EVEX-INT | N/A | APX_F, ADX |
| ADCX r64, r64/m64 | APX-EVEX-INT | N/A | APX_F, ADX |

| | | | |
|---|---|---|---|
| ADCX r32, r32, r32/m32 | APX-EVEX-INT | N/A | APX_F, ADX |
| ADCX r64, r64, r64/m64 | APX-EVEX-INT | N/A | APX_F, ADX |

## 6.3 ADD

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED 00 /r<br>ADD {NF} {ND=0} r8/m8, r8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 00 /r<br>ADD {NF} {ND=1} r8, r8/m8, r8 | F | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 01 /r<br>ADD {NF} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 01 /r<br>ADD {NF} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 01 /r<br>ADD {NF} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 01 /r<br>ADD {NF} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 02 /r<br>ADD {NF} {ND=0} r8, r8/m8 | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 02 /r<br>ADD {NF} {ND=1} r8, r8, r8/m8 | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 03 /r<br>ADD {NF} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 03 /r<br>ADD {NF} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 03 /r<br>ADD {NF} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 03 /r<br>ADD {NF} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /0 ib<br>ADD {NF} {ND=0} r8/m8, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /0 ib<br>ADD {NF} {ND=1} r8, r8/m8, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /0 id<br>ADD {NF} {ND=0} rv/mv, imm32 | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 81 /0 iw/id<br>ADD {NF} {ND=0} rv/mv, imm16/imm32 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /0 id<br>ADD {NF} {ND=1} rv, rv/mv, imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /0 iw/id<br>ADD {NF} {ND=1} rv, rv/mv, imm16/imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /0 ib<br>ADD {NF} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /0 ib<br>ADD {NF} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /0 ib<br>ADD {NF} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /0 ib<br>ADD {NF} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |

### 6.3.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | MODRM.REG(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |
| C | NO-SCALE | MODRM.R/M(rw) | IMM16/IMM32(r) | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM16/IMM32(r) | N/A |
| E | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| F | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | MODRM.REG(r) | N/A |
| G | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |
| H | NO-SCALE | VVVVV(w) | MODRM.REG(r) | MODRM.R/M(r) | N/A |

### 6.3.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.3.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| ADD r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| ADD r8, r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| ADD rv/mv, rv | APX-EVEX-INT | N/A | APX_F |
| ADD rv, rv/mv, rv | APX-EVEX-INT | N/A | APX_F |
| ADD r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| ADD r8, r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| ADD rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| ADD rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| ADD r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| ADD r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| ADD rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| ADD rv, rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| ADD rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| ADD rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |

## 6.4 ADOX

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4.W0 66 /r<br>ADOX {NF=0} {ND=0} r32, r32/m32 | A | V/N.E. | APX_F<br>ADX |
| EVEX.LLZ.F3.MAP4.W1 66 /r<br>ADOX {NF=0} {ND=0} r64, r64/m64 | A | V/N.E. | APX_F<br>ADX |
| EVEX.LLZ.F3.MAP4.W0 66 /r<br>ADOX {NF=0} {ND=1} r32, r32, r32/m32 | B | V/N.E. | APX_F<br>ADX |
| EVEX.LLZ.F3.MAP4.W1 66 /r<br>ADOX {NF=0} {ND=1} r64, r64, r64/m64 | B | V/N.E. | APX_F<br>ADX |

### 6.4.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.REG(r) | MODRM.R/M(r) | N/A |

### 6.4.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.4.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| ADOX r32, r32/m32 | APX-EVEX-INT | N/A | APX_F, ADX |
| ADOX r64, r64/m64 | APX-EVEX-INT | N/A | APX_F, ADX |

| | | | |
|---|---|---|---|
| ADOX r32, r32, r32/m32 | APX-EVEX-INT | N/A | APX_F, ADX |
| ADOX r64, r64, r64/m64 | APX-EVEX-INT | N/A | APX_F, ADX |

## 6.5 AESDEC128KL

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4. DD !(11):rrr:bbb<br>AESDEC128KL {NF=0} {ND=0} xmm1, m384 | A | V/N.E. | APX_F<br>KEYLOCKER |

### 6.5.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |

### 6.5.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.5.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| AESDEC128KL xmm1, m384 | APX-EVEX-KEYLOCKER | N/A | APX_F, KEYLOCKER |

## 6.6 AESDEC256KL

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4. DF !(11):rrr:bbb<br>AESDEC256KL {NF=0} {ND=0} xmm1, m512 | A | V/N.E. | APX_F<br>KEYLOCKER |

### 6.6.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |

### 6.6.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.6.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| AESDEC256KL xmm1, m512 | APX-EVEX-KEYLOCKER | N/A | APX_F, KEYLOCKER |

# 6.7 AESDECWIDE128KL

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4. D8 !(11):001:bbb<br>AESDECWIDE128KL {NF=0} {ND=0} m384 | A | V/N.E. | APX_F<br>KEYLOCKER_WIDE |

## 6.7.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(r) | N/A | N/A | N/A |

## 6.7.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.7.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| AESDECWIDE128KL m384, <xmm0:rw>, <xmm1:rw>, <xmm2:rw>, <xmm3:rw>, <xmm4:rw>, <xmm5:rw>, <xmm6:rw>, <xmm7:rw> | APX-EVEX-KEYLOCKER | N/A | APX_F, KEYLOCKER_WIDE |

# 6.8 AESDECWIDE256KL

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4. D8 !(11):011:bbb<br>AESDECWIDE256KL {NF=0} {ND=0} m512 | A | V/N.E. | APX_F<br>KEYLOCKER_WIDE |

## 6.8.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(r) | N/A | N/A | N/A |

## 6.8.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.8.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| AESDECWIDE256KL m512, <xmm0:rw>, <xmm1:rw>, <xmm2:rw>, <xmm3:rw>, <xmm4:rw>, <xmm5:rw>, <xmm6:rw>, <xmm7:rw> | APX-EVEX-KEYLOCKER | N/A | APX_F, KEYLOCKER_WIDE |

## 6.9 AESENC128KL

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4. DC !(11):rrr:bbb<br>AESENC128KL {NF=0} {ND=0} xmm1, m384 | A | V/N.E. | APX_F<br>KEYLOCKER |

### 6.9.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |

### 6.9.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.9.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| AESENC128KL xmm1, m384 | APX-EVEX-KEYLOCKER | N/A | APX_F, KEYLOCKER |

# 6.10 AESENC256KL

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4. DE !(11):rrr:bbb<br>AESENC256KL {NF=0} {ND=0} xmm1, m512 | A | V/N.E. | APX_F<br>KEYLOCKER |

## 6.10.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |

## 6.10.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.10.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| AESENC256KL xmm1, m512 | APX-EVEX-KEYLOCKER | N/A | APX_F, KEYLOCKER |

# 6.11 AESENCWIDE128KL

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4. D8 !(11):000:bbb<br>AESENCWIDE128KL {NF=0} {ND=0} m384 | A | V/N.E. | APX_F<br>KEYLOCKER_WIDE |

## 6.11.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(r) | N/A | N/A | N/A |

## 6.11.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.11.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| AESENCWIDE128KL m384, <xmm0:rw>, <xmm1:rw>, <xmm2:rw>, <xmm3:rw>, <xmm4:rw>, <xmm5:rw>, <xmm6:rw>, <xmm7:rw> | APX-EVEX-KEYLOCKER | N/A | APX_F, KEYLOCKER_WIDE |

## 6.12 AESENCWIDE256KL

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4. D8 !(11):010:bbb<br>AESENCWIDE256KL {NF=0} {ND=0} m512 | A | V/N.E. | APX_F<br>KEYLOCKER_WIDE |

### 6.12.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(r) | N/A | N/A | N/A |

### 6.12.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.12.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| AESENCWIDE256KL m512, <xmm0:rw>, <xmm1:rw>, <xmm2:rw>, <xmm3:rw>, <xmm4:rw>, <xmm5:rw>, <xmm6:rw>, <xmm7:rw> | APX-EVEX-KEYLOCKER | N/A | APX_F, KEYLOCKER_WIDE |

## 6.13 AND

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED 20 /r AND {NF} {ND=0} r8/m8, r8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 20 /r AND {NF} {ND=1} r8, r8/m8, r8 | F | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 21 /r AND {NF} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 21 /r AND {NF} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 21 /r AND {NF} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 21 /r AND {NF} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 22 /r AND {NF} {ND=0} r8, r8/m8 | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 22 /r AND {NF} {ND=1} r8, r8, r8/m8 | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 23 /r AND {NF} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 23 /r AND {NF} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 23 /r AND {NF} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 23 /r AND {NF} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /4 ib AND {NF} {ND=0} r8/m8, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /4 ib AND {NF} {ND=1} r8, r8/m8, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /4 id AND {NF} {ND=0} rv/mv, imm32 | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 81 /4 iw/id AND {NF} {ND=0} rv/mv, imm16/imm32 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /4 id AND {NF} {ND=1} rv, rv/mv, imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /4 iw/id AND {NF} {ND=1} rv, rv/mv, imm16/imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /4 ib AND {NF} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /4 ib AND {NF} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /4 ib AND {NF} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /4 ib AND {NF} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |

### 6.13.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | MODRM.REG(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |
| C | NO-SCALE | MODRM.R/M(rw) | IMM16/IMM32(r) | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM16/IMM32(r) | N/A |
| E | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| F | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | MODRM.REG(r) | N/A |
| G | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |
| H | NO-SCALE | VVVVV(w) | MODRM.REG(r) | MODRM.R/M(r) | N/A |

### 6.13.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.13.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| AND r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| AND r8, r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| AND rv/mv, rv | APX-EVEX-INT | N/A | APX_F |
| AND rv, rv/mv, rv | APX-EVEX-INT | N/A | APX_F |
| AND r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| AND r8, r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| AND rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| AND rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| AND r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| AND r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| AND rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| AND rv, rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| AND rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| AND rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |

# 6.14 ANDN

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.NP.0F38.W0 F2 /r<br>ANDN {NF} {ND=0} r32, r32, m32/r32 | A | V/N.E. | APX_F<br>BMI1 |
| EVEX.128.NP.0F38.W1 F2 /r<br>ANDN {NF} {ND=0} r64, r64, m64/r64 | A | V/N.E. | APX_F<br>BMI1 |

## 6.14.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | VVVVV(r) | MODRM.R/M(r) | N/A |

## 6.14.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.14.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| ANDN r32, r32, m32/r32 | APX-EVEX-BMI | N/A | APX_F, BMI1 |
| ANDN r64, r64, m64/r64 | APX-EVEX-BMI | N/A | APX_F, BMI1 |

# 6.15 BEXTR

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.NP.0F38.W0 F7 /r<br>BEXTR {NF} {ND=0} r32, m32/r32, r32 | A | V/N.E. | APX_F<br>BMI1 |
| EVEX.128.NP.0F38.W1 F7 /r<br>BEXTR {NF} {ND=0} r64, m64/r64, r64 | A | V/N.E. | APX_F<br>BMI1 |

## 6.15.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | VVVVV(r) | N/A |

## 6.15.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.15.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| BEXTR r32, m32/r32, r32 | APX-EVEX-BMI | N/A | APX_F, BMI1 |
| BEXTR r64, m64/r64, r64 | APX-EVEX-BMI | N/A | APX_F, BMI1 |

## 6.16 BLSI

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.NP.0F38.W0 F3 /3<br>BLSI {NF} {ND=0} r32, m32/r32 | A | V/N.E. | APX_F<br>BMI1 |
| EVEX.128.NP.0F38.W1 F3 /3<br>BLSI {NF} {ND=0} r64, m64/r64 | A | V/N.E. | APX_F<br>BMI1 |

### 6.16.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |

### 6.16.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.16.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| BLSI r32, m32/r32 | APX-EVEX-BMI | N/A | APX_F, BMI1 |
| BLSI r64, m64/r64 | APX-EVEX-BMI | N/A | APX_F, BMI1 |

# 6.17 BLSMSK

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.NP.0F38.W0 F3 /2<br>BLSMSK {NF} {ND=0} r32, m32/r32 | A | V/N.E. | APX_F<br>BMI1 |
| EVEX.128.NP.0F38.W1 F3 /2<br>BLSMSK {NF} {ND=0} r64, m64/r64 | A | V/N.E. | APX_F<br>BMI1 |

## 6.17.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |

## 6.17.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.17.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| BLSMSK r32, m32/r32 | APX-EVEX-BMI | N/A | APX_F, BMI1 |
| BLSMSK r64, m64/r64 | APX-EVEX-BMI | N/A | APX_F, BMI1 |

# 6.18 BLSR

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.NP.0F38.W0 F3 /1<br>BLSR {NF} {ND=0} r32, m32/r32 | A | V/N.E. | APX_F<br>BMI1 |
| EVEX.128.NP.0F38.W1 F3 /1<br>BLSR {NF} {ND=0} r64, m64/r64 | A | V/N.E. | APX_F<br>BMI1 |

## 6.18.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |

## 6.18.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.18.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| BLSR r32, m32/r32 | APX-EVEX-BMI | N/A | APX_F, BMI1 |
| BLSR r64, m64/r64 | APX-EVEX-BMI | N/A | APX_F, BMI1 |

# 6.19 BZHI

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.NP.0F38.W0 F5 /r<br>BZHI {NF} {ND=0} r32, m32/r32, r32 | A | V/N.E. | APX_F<br>BMI2 |
| EVEX.128.NP.0F38.W1 F5 /r<br>BZHI {NF} {ND=0} r64, m64/r64, r64 | A | V/N.E. | APX_F<br>BMI2 |

## 6.19.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | VVVVV(r) | N/A |

## 6.19.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.19.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| BZHI r32, m32/r32, r32 | APX-EVEX-BMI | N/A | APX_F, BMI2 |
| BZHI r64, m64/r64, r64 | APX-EVEX-BMI | N/A | APX_F, BMI2 |

## 6.20 CMOVCC

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE 42 /r<br>CMOVB {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 42 /r<br>CMOVB {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 46 /r<br>CMOVBE {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 46 /r<br>CMOVBE {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4C /r<br>CMOVL {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4C /r<br>CMOVL {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4E /r<br>CMOVLE {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4E /r<br>CMOVLE {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 43 /r<br>CMOVNB {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 43 /r<br>CMOVNB {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 47 /r<br>CMOVNBE {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 47 /r<br>CMOVNBE {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4D /r<br>CMOVNL {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4D /r<br>CMOVNL {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4F /r<br>CMOVNLE {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 4F /r<br>CMOVNLE {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 41 /r<br>CMOVNO {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 41 /r<br>CMOVNO {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4B /r<br>CMOVNP {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4B /r<br>CMOVNP {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 49 /r<br>CMOVNS {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 49 /r<br>CMOVNS {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 45 /r<br>CMOVNZ {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 45 /r<br>CMOVNZ {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 40 /r<br>CMOVO {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 40 /r<br>CMOVO {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4A /r<br>CMOVP {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4A /r<br>CMOVP {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 48 /r<br>CMOVS {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 48 /r<br>CMOVS {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 44 /r<br>CMOVZ {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 44 /r<br>CMOVZ {NF=0} {ND=1} rv, rv, rv/mv | A | V/N.E. | APX_F |

### 6.20.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | VVVVV(w) | MODRM.REG(r) | MODRM.R/M(r) | N/A |

### 6.20.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.20.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| CMOVB rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVBE rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVL rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVLE rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVNB rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVNBE rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVNL rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVNLE rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |

| | | | |
|---|---|---|---|
| CMOVNO rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVNP rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVNS rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVNZ rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVO rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVP rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVS rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| CMOVZ rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |

## 6.21 CMPCCXADD

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.66.0F38.W0 E6 !(11):rrr:bbb<br>CMPBEXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W1 E6 !(11):rrr:bbb<br>CMPBEXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W0 E2 !(11):rrr:bbb<br>CMPBXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W1 E2 !(11):rrr:bbb<br>CMPBXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W0 EE !(11):rrr:bbb<br>CMPLEXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W1 EE !(11):rrr:bbb<br>CMPLEXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W0 EC !(11):rrr:bbb<br>CMPLXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W1 EC !(11):rrr:bbb<br>CMPLXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W0 E7 !(11):rrr:bbb<br>CMPNBEXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W1 E7 !(11):rrr:bbb<br>CMPNBEXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W0 E3 !(11):rrr:bbb<br>CMPNBXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W1 E3 !(11):rrr:bbb<br>CMPNBXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W0 EF !(11):rrr:bbb<br>CMPNLEXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W1 EF !(11):rrr:bbb<br>CMPNLEXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F<br>CMPCCXADD |
| EVEX.128.66.0F38.W0 ED !(11):rrr:bbb<br>CMPNLXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F<br>CMPCCXADD |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.66.0F38.W1 ED !(11):rrr:bbb CMPNLXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W0 E1 !(11):rrr:bbb CMPNOXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W1 E1 !(11):rrr:bbb CMPNOXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W0 EB !(11):rrr:bbb CMPNPXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W1 EB !(11):rrr:bbb CMPNPXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W0 E9 !(11):rrr:bbb CMPNSXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W1 E9 !(11):rrr:bbb CMPNSXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W0 E5 !(11):rrr:bbb CMPNZXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W1 E5 !(11):rrr:bbb CMPNZXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W0 E0 !(11):rrr:bbb CMPOXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W1 E0 !(11):rrr:bbb CMPOXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W0 EA !(11):rrr:bbb CMPPXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W1 EA !(11):rrr:bbb CMPPXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W0 E8 !(11):rrr:bbb CMPSXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W1 E8 !(11):rrr:bbb CMPSXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F CMPCCXADD |
| EVEX.128.66.0F38.W0 E4 !(11):rrr:bbb CMPZXADD {NF=0} {ND=0} m32, r32, r32 | A | V/N.E. | APX_F CMPCCXADD |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.66.0F38.W1 E4 !(11):rrr:bbb<br>CMPZXADD {NF=0} {ND=0} m64, r64, r64 | A | V/N.E. | APX_F<br>CMPCCXADD |

### 6.21.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | MODRM.REG(rw) | VVVVV(r) | N/A |

### 6.21.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.21.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| CMPBEXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPBEXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPBXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPBXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPLEXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPLEXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPLXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPLXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |

| | | | |
|---|---|---|---|
| CMPNBEXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNBEXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNBXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNBXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNLEXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNLEXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNLXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNLXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNOXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNOXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNPXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNPXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNSXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNSXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNZXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPNZXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPOXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPOXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPPXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPPXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPSXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPSXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |
| CMPZXADD m32, r32, r32 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |

| | | | |
|---|---|---|---|
| CMPZXADD m64, r64, r64 | APX-EVEX-CMPCCXADD | N/A | APX_F, CMPCCXADD |

## 6.22 CRC32

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE F0 /r<br>CRC32 {NF=0} {ND=0} ry, r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F1 /r<br>CRC32 {NF=0} {ND=0} ry, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F1 /r<br>CRC32 {NF=0} {ND=0} ry, rv/mv | A | V/N.E. | APX_F |

### 6.22.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |

### 6.22.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.22.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| CRC32 ry, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| CRC32 ry, rv/mv | APX-EVEX-INT | N/A | APX_F |

## 6.23 DEC

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED FE /1<br>DEC {NF} {ND=0} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED FE /1<br>DEC {NF} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE FF /1<br>DEC {NF} {ND=0} rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE FF /1<br>DEC {NF} {ND=0} rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE FF /1<br>DEC {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE FF /1<br>DEC {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |

### 6.23.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | N/A | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |

### 6.23.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.23.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|

| | | | |
|---|---|---|---|
| DEC r8/m8 | APX-EVEX-INT | N/A | APX_F |
| DEC r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| DEC rv/mv | APX-EVEX-INT | N/A | APX_F |
| DEC rv, rv/mv | APX-EVEX-INT | N/A | APX_F |

## 6.24 DIV

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED F6 /6<br>DIV {NF} {ND=0} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /6<br>DIV {NF} {ND=0} rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /6<br>DIV {NF} {ND=0} rv/mv | A | V/N.E. | APX_F |

### 6.24.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(r) | N/A | N/A | N/A |

### 6.24.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.24.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| DIV r8/m8, <ax:rw:supp> | APX-EVEX-INT | N/A | APX_F |
| DIV rv/mv, <orax:rw:supp>, <ordx:rw:supp> | APX-EVEX-INT | N/A | APX_F |

# 6.25 ENCODEKEY128

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4. DA 11:rrr:bbb<br>ENCODEKEY128 {NF=0} {ND=0} r32, r32 | A | V/N.E. | APX_F<br>KEYLOCKER |

## 6.25.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | N/A | N/A |

## 6.25.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.25.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| ENCODEKEY128 r32, r32, <xmm0:rw>, <xmm1:w>, <xmm2:w>, <xmm4:w>, <xmm5:w>, <xmm6:w> | APX-EVEX-KEYLOCKER | N/A | APX_F, KEYLOCKER |

# 6.26 ENCODEKEY256

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4. DB 11:rrr:bbb<br>ENCODEKEY256 {NF=0} {ND=0} r32, r32 | A | V/N.E. | APX_F<br>KEYLOCKER |

## 6.26.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | N/A | N/A |

## 6.26.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.26.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| ENCODEKEY256 r32, r32, <xmm0:rw>, <xmm1:rw>, <xmm2:w>, <xmm3:w>, <xmm4:w>, <xmm5:w>, <xmm6:w> | APX-EVEX-KEYLOCKER | N/A | APX_F, KEYLOCKER |

# 6.27 ENQCMD

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F2.MAP4. F8 !(11):rrr:bbb<br>ENQCMD {NF=0} {ND=0} ra, m512 | A | V/N.E. | APX_F<br>ENQCMD |

## 6.27.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(r) | MODRM.R/M(r) | N/A | N/A |

## 6.27.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.27.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| ENQCMD ra, m512 | APX-EVEX-ENQCMD | N/A | APX_F, ENQCMD |

# 6.28 ENQCMDS

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4. F8 !(11):rrr:bbb<br>ENQCMDS {NF=0} {ND=0} ra, m512 | A | V/N.E. | APX_F<br>ENQCMD |

## 6.28.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(r) | MODRM.R/M(r) | N/A | N/A |

## 6.28.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.28.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| ENQCMDS ra, m512 | APX-EVEX-ENQCMD | N/A | APX_F, ENQCMD |

# 6.29 IDIV

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED F6 /7<br>IDIV {NF} {ND=0} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /7<br>IDIV {NF} {ND=0} rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /7<br>IDIV {NF} {ND=0} rv/mv | A | V/N.E. | APX_F |

## 6.29.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(r) | N/A | N/A | N/A |

## 6.29.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.29.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| IDIV r8/m8, <ax:rw:supp> | APX-EVEX-INT | N/A | APX_F |
| IDIV rv/mv, <orax:rw:supp>, <ordx:rw:supp> | APX-EVEX-INT | N/A | APX_F |

## 6.30 IMUL

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE 69 /r id<br>IMUL {NF} {ND=ZU} rv, rv/mv, imm32 | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 69 /r iw/id<br>IMUL {NF} {ND=ZU} rv, rv/mv, imm16/imm32 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 6B /r ib<br>IMUL {NF} {ND=ZU} rv, rv/mv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 6B /r ib<br>IMUL {NF} {ND=ZU} rv, rv/mv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE AF /r<br>IMUL {NF} {ND=0} rv, rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE AF /r<br>IMUL {NF} {ND=0} rv, rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE AF /r<br>IMUL {NF} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE AF /r<br>IMUL {NF} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /5<br>IMUL {NF} {ND=0} r8/m8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /5<br>IMUL {NF} {ND=0} rv/mv | E | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /5<br>IMUL {NF} {ND=0} rv/mv | E | V/N.E. | APX_F |

### 6.30.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | IMM16/IMM32(r) | N/A |
| B | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | IMM8(r) | N/A |
| C | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.REG(r) | MODRM.R/M(r) | N/A |
| E | NO-SCALE | MODRM.R/M(r) | N/A | N/A | N/A |

### 6.30.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.30.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| IMUL rv, rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| IMUL rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| IMUL rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| IMUL rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| IMUL r8/m8, <al:r:supp>, <ax:w:supp> | APX-EVEX-INT | N/A | APX_F |
| IMUL rv/mv, <orax:rw:supp>, <ordx:w:supp> | APX-EVEX-INT | N/A | APX_F |

# 6.31 INC

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED FE /0<br>INC {NF} {ND=0} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED FE /0<br>INC {NF} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE FF /0<br>INC {NF} {ND=0} rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE FF /0<br>INC {NF} {ND=0} rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE FF /0<br>INC {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE FF /0<br>INC {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |

## 6.31.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | N/A | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |

## 6.31.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.31.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|

| | | | |
|---|---|---|---|
| INC r8/m8 | APX-EVEX-INT | N/A | APX_F |
| INC r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| INC rv/mv | APX-EVEX-INT | N/A | APX_F |
| INC rv, rv/mv | APX-EVEX-INT | N/A | APX_F |

# 6.32 INVEPT

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4.IGNORED F0 !(11):rrr:bbb<br>INVEPT {NF=0} {ND=0} r64, m128 | A | V/N.E. | APX_F<br>VMX |

## 6.32.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(r) | MODRM.R/M(r) | N/A | N/A |

## 6.32.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.32.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| INVEPT r64, m128 | APX-EVEX-INVEPT | N/A | APX_F, VMX |

# 6.33 INVPCID

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4.IGNORED F2 !(11):rrr:bbb<br>INVPCID {NF=0} {ND=0} r64, m128 | A | V/N.E. | APX_F<br>INVPCID |

## 6.33.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(r) | MODRM.R/M(r) | N/A | N/A |

## 6.33.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.33.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| INVPCID r64, m128 | APX-EVEX-INVPCID | N/A | APX_F, INVPCID |

# 6.34 INVVPID

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F3.MAP4.IGNORED F1 !(11):rrr:bbb<br>INVVPID {NF=0} {ND=0} r64, m128 | A | V/N.E. | APX_F<br>VMX |

## 6.34.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(r) | MODRM.R/M(r) | N/A | N/A |

## 6.34.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.34.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| INVVPID r64, m128 | APX-EVEX-INVVPID | N/A | APX_F, VMX |

## 6.35 KMOVB

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.66.0F.W0 90 /r<br>KMOVB {NF=0} {ND=0} k1, k2/m8 | A | V/N.E. | APX_F<br>(AVX512DQ OR AVX10.1) |
| EVEX.128.66.0F.W0 92 11:rrr:bbb<br>KMOVB {NF=0} {ND=0} k1, r32 | A | V/N.E. | APX_F<br>(AVX512DQ OR AVX10.1) |
| EVEX.128.66.0F.W0 93 11:rrr:bbb<br>KMOVB {NF=0} {ND=0} r32, k1 | A | V/N.E. | APX_F<br>(AVX512DQ OR AVX10.1) |
| EVEX.128.66.0F.W0 91 !(11):rrr:bbb<br>KMOVB {NF=0} {ND=0} m8, k1 | B | V/N.E. | APX_F<br>(AVX512DQ OR AVX10.1) |

### 6.35.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | N/A | N/A |
| B | NO-SCALE | MODRM.R/M(w) | MODRM.REG(r) | N/A | N/A |

### 6.35.2 Description

**Note:**

For instructions with a CPUID feature flag specifying AVX10, the programmer must check the available vector options on the processor at run-time via the CPU_SUPPORTED_VECTOR_LENGTHS field in Converged Vector ISA Leaf 0x24. This field enumerates the maximum supported vector width and as such will determine the set of instructions available to the programmer listed in the above opcode table.

**Note:**

For opmask instructions with a CPUID feature flag specifying AVX10, the programmer must check the available vector options on the processor at run-time via CPUID Leaf 0x24, the Intel AVX10 Converged Vector ISA Leaf. This leaf enumerates the maximum supported vector width and as such will determine the set of supported opmask instructions available to the programmer listed in the above opcode table. Quadword opmask instructions will only be supported on processors supporting vector lengths of 512 bits.

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.35.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| KMOVB k1, k2/m8 | APX-EVEX-KMOV | N/A | APX_F, (AVX512DQ OR AVX10.1) |
| KMOVB k1, r32 | APX-EVEX-KMOV | N/A | APX_F, (AVX512DQ OR AVX10.1) |
| KMOVB r32, k1 | APX-EVEX-KMOV | N/A | APX_F, (AVX512DQ OR AVX10.1) |
| KMOVB m8, k1 | APX-EVEX-KMOV | N/A | APX_F, (AVX512DQ OR AVX10.1) |

# 6.36 KMOVD

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.F2.0F.W0 93 11:rrr:bbb<br>KMOVD {NF=0} {ND=0} r32, k1 | A | V/N.E. | APX_F<br>(AVX512BW OR AVX10.1) |
| EVEX.128.66.0F.W1 90 /r<br>KMOVD {NF=0} {ND=0} k1, k2/m32 | A | V/N.E. | APX_F<br>(AVX512BW OR AVX10.1) |
| EVEX.128.F2.0F.W0 92 11:rrr:bbb<br>KMOVD {NF=0} {ND=0} k1, r32 | A | V/N.E. | APX_F<br>(AVX512BW OR AVX10.1) |
| EVEX.128.66.0F.W1 91 !(11):rrr:bbb<br>KMOVD {NF=0} {ND=0} m32, k1 | B | V/N.E. | APX_F<br>(AVX512BW OR AVX10.1) |

## 6.36.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | N/A | N/A |
| B | NO-SCALE | MODRM.R/M(w) | MODRM.REG(r) | N/A | N/A |

## 6.36.2 Description

**Note:**

For instructions with a CPUID feature flag specifying AVX10, the programmer must check the available vector options on the processor at run-time via the CPU_SUPPORTED_VECTOR_LENGTHS field in Converged Vector ISA Leaf 0x24. This field enumerates the maximum supported vector width and as such will determine the set of instructions available to the programmer listed in the above opcode table.

**Note:**

For opmask instructions with a CPUID feature flag specifying AVX10, the programmer must check the available vector options on the processor at run-time via CPUID Leaf 0x24, the Intel AVX10 Converged Vector ISA Leaf. This leaf enumerates the maximum supported vector width and as such will determine the set of supported opmask instructions available to the programmer listed in the above opcode table. Quadword opmask instructions will only be supported on processors supporting vector lengths of 512 bits.

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.36.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| KMOVD r32, k1 | APX-EVEX-KMOV | N/A | APX_F, (AVX512BW OR AVX10.1) |
| KMOVD k1, k2/m32 | APX-EVEX-KMOV | N/A | APX_F, (AVX512BW OR AVX10.1) |
| KMOVD k1, r32 | APX-EVEX-KMOV | N/A | APX_F, (AVX512BW OR AVX10.1) |
| KMOVD m32, k1 | APX-EVEX-KMOV | N/A | APX_F, (AVX512BW OR AVX10.1) |

# 6.37 KMOVQ

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.NP.0F.W1 90 /r<br>KMOVQ {NF=0} {ND=0} k1, k2/m64 | A | V/N.E. | APX_F<br>(AVX512BW OR AVX10.1) |
| EVEX.128.F2.0F.W1 92 11:rrr:bbb<br>KMOVQ {NF=0} {ND=0} k1, r64 | A | V/N.E. | APX_F<br>(AVX512BW OR AVX10.1) |
| EVEX.128.F2.0F.W1 93 11:rrr:bbb<br>KMOVQ {NF=0} {ND=0} r64, k1 | A | V/N.E. | APX_F<br>(AVX512BW OR AVX10.1) |
| EVEX.128.NP.0F.W1 91 !(11):rrr:bbb<br>KMOVQ {NF=0} {ND=0} m64, k1 | B | V/N.E. | APX_F<br>(AVX512BW OR AVX10.1) |

## 6.37.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | N/A | N/A |
| B | NO-SCALE | MODRM.R/M(w) | MODRM.REG(r) | N/A | N/A |

## 6.37.2 Description

**Note:**

For instructions with a CPUID feature flag specifying AVX10, the programmer must check the available vector options on the processor at run-time via the CPU_SUPPORTED_VECTOR_LENGTHS field in Converged Vector ISA Leaf 0x24. This field enumerates the maximum supported vector width and as such will determine the set of instructions available to the programmer listed in the above opcode table.

**Note:**

For opmask instructions with a CPUID feature flag specifying AVX10, the programmer must check the available vector options on the processor at run-time via CPUID Leaf 0x24, the Intel AVX10 Converged Vector ISA Leaf. This leaf enumerates the maximum supported vector width and as such will determine the set of supported opmask instructions available to the programmer listed in the above opcode table. Quadword opmask instructions will only be supported on processors supporting vector lengths of 512 bits.

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.37.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| KMOVQ k1, k2/m64 | APX-EVEX-KMOV | N/A | APX_F, (AVX512BW OR AVX10.1) |
| KMOVQ k1, r64 | APX-EVEX-KMOV | N/A | APX_F, (AVX512BW OR AVX10.1) |
| KMOVQ r64, k1 | APX-EVEX-KMOV | N/A | APX_F, (AVX512BW OR AVX10.1) |
| KMOVQ m64, k1 | APX-EVEX-KMOV | N/A | APX_F, (AVX512BW OR AVX10.1) |

# 6.38 KMOVW

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.NP.0F.W0 90 /r<br>KMOVW {NF=0} {ND=0} k1, k2/m16 | A | V/N.E. | APX_F<br>(AVX512F OR AVX10.1) |
| EVEX.128.NP.0F.W0 92 11:rrr:bbb<br>KMOVW {NF=0} {ND=0} k1, r32 | A | V/N.E. | APX_F<br>(AVX512F OR AVX10.1) |
| EVEX.128.NP.0F.W0 93 11:rrr:bbb<br>KMOVW {NF=0} {ND=0} r32, k1 | A | V/N.E. | APX_F<br>(AVX512F OR AVX10.1) |
| EVEX.128.NP.0F.W0 91 !(11):rrr:bbb<br>KMOVW {NF=0} {ND=0} m16, k1 | B | V/N.E. | APX_F<br>(AVX512F OR AVX10.1) |

## 6.38.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | N/A | N/A |
| B | NO-SCALE | MODRM.R/M(w) | MODRM.REG(r) | N/A | N/A |

## 6.38.2 Description

**Note:**

For instructions with a CPUID feature flag specifying AVX10, the programmer must check the available vector options on the processor at run-time via the CPU_SUPPORTED_VECTOR_LENGTHS field in Converged Vector ISA Leaf 0x24. This field enumerates the maximum supported vector width and as such will determine the set of instructions available to the programmer listed in the above opcode table.

**Note:**

For opmask instructions with a CPUID feature flag specifying AVX10, the programmer must check the available vector options on the processor at run-time via CPUID Leaf 0x24, the Intel AVX10 Converged Vector ISA Leaf. This leaf enumerates the maximum supported vector width and as such will determine the set of supported opmask instructions available to the programmer listed in the above opcode table. Quadword opmask instructions will only be supported on processors supporting vector lengths of 512 bits.

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.38.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| KMOVW k1, k2/m16 | APX-EVEX-KMOV | N/A | APX_F, (AVX512F OR AVX10.1) |
| KMOVW k1, r32 | APX-EVEX-KMOV | N/A | APX_F, (AVX512F OR AVX10.1) |
| KMOVW r32, k1 | APX-EVEX-KMOV | N/A | APX_F, (AVX512F OR AVX10.1) |
| KMOVW m16, k1 | APX-EVEX-KMOV | N/A | APX_F, (AVX512F OR AVX10.1) |

# 6.39 LDTILECFG

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.NP.0F38.W0 49 !(11):000:bbb<br>LDTILECFG {NF=0} {ND=0} m512 | A | V/N.E. | APX_F<br>AMX-TILE |

## 6.39.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | | MODRM.R/M(r) | N/A | N/A | N/A |

## 6.39.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.39.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| LDTILECFG m512 | AMX-E1-EVEX | N/A | APX_F, AMX-TILE |

# 6.40 LZCNT

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE F5 /r<br>LZCNT {NF} {ND=0} rv, rv/mv | A | V/N.E. | APX_F<br>LZCNT |
| EVEX.LLZ.66.MAP4.SCALABLE F5 /r<br>LZCNT {NF} {ND=0} rv, rv/mv | A | V/N.E. | APX_F<br>LZCNT |

## 6.40.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | N/A | N/A |

## 6.40.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.40.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| LZCNT rv, rv/mv | APX-EVEX-INT | N/A | APX_F, LZCNT |

# 6.41 MOVBE

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE 60 /r<br>MOVBE {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F<br>MOVBE |
| EVEX.LLZ.66.MAP4.SCALABLE 60 /r<br>MOVBE {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F<br>MOVBE |
| EVEX.LLZ.NP.MAP4.SCALABLE 61 /r<br>MOVBE {NF=0} {ND=0} rv/mv, rv | B | V/N.E. | APX_F<br>MOVBE |
| EVEX.LLZ.66.MAP4.SCALABLE 61 /r<br>MOVBE {NF=0} {ND=0} rv/mv, rv | B | V/N.E. | APX_F<br>MOVBE |

## 6.41.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | N/A | N/A |
| B | NO-SCALE | MODRM.R/M(w) | MODRM.REG(r) | N/A | N/A |

## 6.41.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.41.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| MOVBE rv, rv/mv | APX-EVEX-INT | N/A | APX_F, MOVBE |
| MOVBE rv/mv, rv | APX-EVEX-INT | N/A | APX_F, MOVBE |

# 6.42 MOVDIR64B

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4. F8 !(11):rrr:bbb<br>MOVDIR64B {NF=0} {ND=0} ra, m512 | A | V/N.E. | APX_F<br>MOVDIR |

## 6.42.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(r) | MODRM.R/M(r) | N/A | N/A |

## 6.42.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.42.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| MOVDIR64B ra, m512 | APX-EVEX-INT | N/A | APX_F, MOVDIR |

# 6.43 MOVDIRI

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4. F9 !(11):rrr:bbb<br>MOVDIRI {NF=0} {ND=0} my, ry | A | V/N.E. | APX_F<br>MOVDIR |

## 6.43.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(w) | MODRM.REG(r) | N/A | N/A |

## 6.43.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.43.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| MOVDIRI my, ry | APX-EVEX-INT | N/A | APX_F, MOVDIR |

## 6.44 MUL

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED F6 /4<br>MUL {NF} {ND=0} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /4<br>MUL {NF} {ND=0} rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /4<br>MUL {NF} {ND=0} rv/mv | A | V/N.E. | APX_F |

### 6.44.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(r) | N/A | N/A | N/A |

### 6.44.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.44.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| MUL r8/m8, <al:r:supp>, <ax:w:supp> | APX-EVEX-INT | N/A | APX_F |
| MUL rv/mv, <orax:rw:supp>, <ordx:w:supp> | APX-EVEX-INT | N/A | APX_F |

# 6.45 MULX

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.F2.0F38.W0 F6 /r<br>MULX {NF=0} {ND=0} r32, r32, m32/r32 | A | V/N.E. | APX_F<br>BMI2 |
| EVEX.128.F2.0F38.W1 F6 /r<br>MULX {NF=0} {ND=0} r64, r64, m64/r64 | A | V/N.E. | APX_F<br>BMI2 |

## 6.45.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | VVVVV(w) | MODRM.R/M(r) | N/A |

## 6.45.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.45.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| MULX r32, r32, m32/r32, <edx:r:supp> | APX-EVEX-BMI | N/A | APX_F, BMI2 |
| MULX r64, r64, m64/r64, <rdx:r:supp> | APX-EVEX-BMI | N/A | APX_F, BMI2 |

# 6.46 NEG

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED F6 /3 NEG {NF} {ND=0} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /3 NEG {NF} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /3 NEG {NF} {ND=0} rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /3 NEG {NF} {ND=0} rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /3 NEG {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /3 NEG {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |

## 6.46.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | N/A | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |

## 6.46.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.46.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|

| | | | |
|---|---|---|---|
| NEG r8/m8 | APX-EVEX-INT | N/A | APX_F |
| NEG r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| NEG rv/mv | APX-EVEX-INT | N/A | APX_F |
| NEG rv, rv/mv | APX-EVEX-INT | N/A | APX_F |

## 6.47 NOT

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED F6 /2<br>NOT {NF=0} {ND=0} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /2<br>NOT {NF=0} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /2<br>NOT {NF=0} {ND=0} rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /2<br>NOT {NF=0} {ND=0} rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /2<br>NOT {NF=0} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /2<br>NOT {NF=0} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |

### 6.47.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | N/A | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |

### 6.47.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.47.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|

| | | | |
|---|---|---|---|
| NOT r8/m8 | APX-EVEX-INT | N/A | APX_F |
| NOT r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| NOT rv/mv | APX-EVEX-INT | N/A | APX_F |
| NOT rv, rv/mv | APX-EVEX-INT | N/A | APX_F |

## 6.48 OR

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED 0A /r<br>OR {NF} {ND=0} r8, r8/m8 | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 0A /r<br>OR {NF} {ND=1} r8, r8, r8/m8 | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 0B /r<br>OR {NF} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 0B /r<br>OR {NF} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 0B /r<br>OR {NF} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 0B /r<br>OR {NF} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 08 /r<br>OR {NF} {ND=0} r8/m8, r8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 08 /r<br>OR {NF} {ND=1} r8, r8/m8, r8 | F | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /1 ib<br>OR {NF} {ND=0} r8/m8, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /1 ib<br>OR {NF} {ND=1} r8, r8/m8, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /1 id<br>OR {NF} {ND=0} rv/mv, imm32 | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /1 iw/id<br>OR {NF} {ND=0} rv/mv, imm16/imm32 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /1 id<br>OR {NF} {ND=1} rv, rv/mv, imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /1 iw/id<br>OR {NF} {ND=1} rv, rv/mv, imm16/imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /1 ib<br>OR {NF} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 83 /1 ib<br>OR {NF} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /1 ib<br>OR {NF} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /1 ib<br>OR {NF} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 09 /r<br>OR {NF} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 09 /r<br>OR {NF} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 09 /r<br>OR {NF} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 09 /r<br>OR {NF} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |

### 6.48.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | MODRM.REG(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |
| C | NO-SCALE | MODRM.R/M(rw) | IMM16/IMM32(r) | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM16/IMM32(r) | N/A |
| E | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| F | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | MODRM.REG(r) | N/A |
| G | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |
| H | NO-SCALE | VVVVV(w) | MODRM.REG(r) | MODRM.R/M(r) | N/A |

### 6.48.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.48.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| OR r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| OR r8, r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| OR rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| OR rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| OR r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| OR r8, r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| OR r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| OR r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| OR rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| OR rv, rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| OR rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| OR rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| OR rv/mv, rv | APX-EVEX-INT | N/A | APX_F |
| OR rv, rv/mv, rv | APX-EVEX-INT | N/A | APX_F |

# 6.49 PDEP

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.F2.0F38.W0 F5 /r<br>PDEP {NF=0} {ND=0} r32, r32, m32/r32 | A | V/N.E. | APX_F<br>BMI2 |
| EVEX.128.F2.0F38.W1 F5 /r<br>PDEP {NF=0} {ND=0} r64, r64, m64/r64 | A | V/N.E. | APX_F<br>BMI2 |

## 6.49.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | VVVVV(r) | MODRM.R/M(r) | N/A |

## 6.49.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.49.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| PDEP r32, r32, m32/r32 | APX-EVEX-BMI | N/A | APX_F, BMI2 |
| PDEP r64, r64, m64/r64 | APX-EVEX-BMI | N/A | APX_F, BMI2 |

# 6.50 PEXT

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.F3.0F38.W0 F5 /r<br>PEXT {NF=0} {ND=0} r32, r32, m32/r32 | A | V/N.E. | APX_F<br>BMI2 |
| EVEX.128.F3.0F38.W1 F5 /r<br>PEXT {NF=0} {ND=0} r64, r64, m64/r64 | A | V/N.E. | APX_F<br>BMI2 |

## 6.50.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | VVVVV(r) | MODRM.R/M(r) | N/A |

## 6.50.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.50.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| PEXT r32, r32, m32/r32 | APX-EVEX-BMI | N/A | APX_F, BMI2 |
| PEXT r64, r64, m64/r64 | APX-EVEX-BMI | N/A | APX_F, BMI2 |

# 6.51 POPCNT

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE 88 /r<br>POPCNT {NF} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 88 /r<br>POPCNT {NF} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |

## 6.51.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | N/A | N/A |

## 6.51.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.51.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| POPCNT rv, rv/mv | APX-EVEX-INT | N/A | APX_F |

## 6.52 RCL

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED C0 /2 ib<br>RCL {NF=0} {ND=0} r8/m8, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED C0 /2 ib<br>RCL {NF=0} {ND=1} r8, r8/m8, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /2 ib<br>RCL {NF=0} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /2 ib<br>RCL {NF=0} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /2 ib<br>RCL {NF=0} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /2 ib<br>RCL {NF=0} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /2<br>RCL {NF=0} {ND=0} r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /2<br>RCL {NF=0} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /2<br>RCL {NF=0} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /2<br>RCL {NF=0} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /2<br>RCL {NF=0} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /2<br>RCL {NF=0} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /2<br>RCL {NF=0} {ND=0} r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /2<br>RCL {NF=0} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /2<br>RCL {NF=0} {ND=0} rv/mv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE D3 /2<br>RCL {NF=0} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /2<br>RCL {NF=0} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D3 /2<br>RCL {NF=0} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |

### 6.52.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |
| C | NO-SCALE | MODRM.R/M(rw) | N/A | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |

### 6.52.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.52.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| RCL r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| RCL r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| RCL rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| RCL rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |

| | | | |
|---|---|---|---|
| RCL r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCL r8, r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCL rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCL rv, rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCL r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCL r8, r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCL rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCL rv, rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |

## 6.53 RCR

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED C0 /3 ib<br>RCR {NF=0} {ND=0} r8/m8, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED C0 /3 ib<br>RCR {NF=0} {ND=1} r8, r8/m8, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /3 ib<br>RCR {NF=0} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /3 ib<br>RCR {NF=0} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /3 ib<br>RCR {NF=0} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /3 ib<br>RCR {NF=0} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /3<br>RCR {NF=0} {ND=0} r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /3<br>RCR {NF=0} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /3<br>RCR {NF=0} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /3<br>RCR {NF=0} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /3<br>RCR {NF=0} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /3<br>RCR {NF=0} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /3<br>RCR {NF=0} {ND=0} r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /3<br>RCR {NF=0} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /3<br>RCR {NF=0} {ND=0} rv/mv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE D3 /3 RCR {NF=0} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /3 RCR {NF=0} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D3 /3 RCR {NF=0} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |

### 6.53.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |
| C | NO-SCALE | MODRM.R/M(rw) | N/A | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |

### 6.53.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.53.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| RCR r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| RCR r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| RCR rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| RCR rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |

| | | | |
|---|---|---|---|
| RCR r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCR r8, r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCR rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCR rv, rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCR r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCR r8, r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCR rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| RCR rv, rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |

## 6.54 ROL

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED C0 /0 ib<br>ROL {NF} {ND=0} r8/m8, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED C0 /0 ib<br>ROL {NF} {ND=1} r8, r8/m8, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /0 ib<br>ROL {NF} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /0 ib<br>ROL {NF} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /0 ib<br>ROL {NF} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /0 ib<br>ROL {NF} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /0<br>ROL {NF} {ND=0} r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /0<br>ROL {NF} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /0<br>ROL {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /0<br>ROL {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /0<br>ROL {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /0<br>ROL {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /0<br>ROL {NF} {ND=0} r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /0<br>ROL {NF} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /0<br>ROL {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE D3 /0 ROL {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /0 ROL {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D3 /0 ROL {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |

### 6.54.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |
| C | NO-SCALE | MODRM.R/M(rw) | N/A | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |

### 6.54.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.54.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| ROL r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| ROL r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| ROL rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| ROL rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |

| | | | |
|---|---|---|---|
| ROL r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROL r8, r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROL rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROL rv, rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROL r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROL r8, r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROL rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROL rv, rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |

## 6.55 ROR

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED C0 /1 ib<br>ROR {NF} {ND=0} r8/m8, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED C0 /1 ib<br>ROR {NF} {ND=1} r8, r8/m8, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /1 ib<br>ROR {NF} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /1 ib<br>ROR {NF} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /1 ib<br>ROR {NF} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /1 ib<br>ROR {NF} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /1<br>ROR {NF} {ND=0} r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /1<br>ROR {NF} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /1<br>ROR {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /1<br>ROR {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /1<br>ROR {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /1<br>ROR {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /1<br>ROR {NF} {ND=0} r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /1<br>ROR {NF} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /1<br>ROR {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE D3 /1<br>ROR {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /1<br>ROR {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D3 /1<br>ROR {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |

### 6.55.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |
| C | NO-SCALE | MODRM.R/M(rw) | N/A | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |

### 6.55.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.55.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| ROR r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| ROR r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| ROR rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| ROR rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |

| | | | |
|---|---|---|---|
| ROR r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROR r8, r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROR rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROR rv, rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROR r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROR r8, r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROR rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| ROR rv, rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |

# 6.56 RORX

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.F2.0F3A.W0 F0 /r /ib<br>RORX {NF=0} {ND=0} r32, m32/r32, imm8 | A | V/N.E. | APX_F<br>BMI2 |
| EVEX.128.F2.0F3A.W1 F0 /r /ib<br>RORX {NF=0} {ND=0} r64, m64/r64, imm8 | A | V/N.E. | APX_F<br>BMI2 |

## 6.56.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | IMM8(r) | N/A |

## 6.56.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.56.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| RORX r32, m32/r32, imm8 | APX-EVEX-BMI | N/A | APX_F, BMI2 |
| RORX r64, m64/r64, imm8 | APX-EVEX-BMI | N/A | APX_F, BMI2 |

## 6.57 SAR

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED C0 /7 ib<br>SAR {NF} {ND=0} r8/m8, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED C0 /7 ib<br>SAR {NF} {ND=1} r8, r8/m8, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /7 ib<br>SAR {NF} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /7 ib<br>SAR {NF} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /7 ib<br>SAR {NF} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /7 ib<br>SAR {NF} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /7<br>SAR {NF} {ND=0} r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /7<br>SAR {NF} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /7<br>SAR {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /7<br>SAR {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /7<br>SAR {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /7<br>SAR {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /7<br>SAR {NF} {ND=0} r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /7<br>SAR {NF} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /7<br>SAR {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE D3 /7 SAR {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /7 SAR {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D3 /7 SAR {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |

### 6.57.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |
| C | NO-SCALE | MODRM.R/M(rw) | N/A | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |

### 6.57.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.57.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SAR r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| SAR r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| SAR rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| SAR rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |

| | | | |
|---|---|---|---|
| SAR r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SAR r8, r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SAR rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SAR rv, rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SAR r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SAR r8, r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SAR rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SAR rv, rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |

# 6.58 SARX

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.F3.0F38.W0 F7 /r<br>SARX {NF=0} {ND=0} r32, m32/r32, r32 | A | V/N.E. | APX_F<br>BMI2 |
| EVEX.128.F3.0F38.W1 F7 /r<br>SARX {NF=0} {ND=0} r64, m64/r64, r64 | A | V/N.E. | APX_F<br>BMI2 |

## 6.58.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | VVVVV(r) | N/A |

## 6.58.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.58.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SARX r32, m32/r32, r32 | APX-EVEX-BMI | N/A | APX_F, BMI2 |
| SARX r64, m64/r64, r64 | APX-EVEX-BMI | N/A | APX_F, BMI2 |

## 6.59 SBB

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED 18 /r<br>SBB {NF=0} {ND=0} r8/m8, r8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 18 /r<br>SBB {NF=0} {ND=1} r8, r8/m8, r8 | F | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 19 /r<br>SBB {NF=0} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 19 /r<br>SBB {NF=0} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 19 /r<br>SBB {NF=0} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 19 /r<br>SBB {NF=0} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 1A /r<br>SBB {NF=0} {ND=0} r8, r8/m8 | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 1A /r<br>SBB {NF=0} {ND=1} r8, r8, r8/m8 | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 1B /r<br>SBB {NF=0} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 1B /r<br>SBB {NF=0} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 1B /r<br>SBB {NF=0} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 1B /r<br>SBB {NF=0} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /3 ib<br>SBB {NF=0} {ND=0} r8/m8, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /3 ib<br>SBB {NF=0} {ND=1} r8, r8/m8, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /3 id<br>SBB {NF=0} {ND=0} rv/mv, imm32 | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 81 /3 iw/id<br>SBB {NF=0} {ND=0} rv/mv, imm16/imm32 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /3 id<br>SBB {NF=0} {ND=1} rv, rv/mv, imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /3 iw/id<br>SBB {NF=0} {ND=1} rv, rv/mv, imm16/imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /3 ib<br>SBB {NF=0} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /3 ib<br>SBB {NF=0} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /3 ib<br>SBB {NF=0} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /3 ib<br>SBB {NF=0} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |

### 6.59.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | MODRM.REG(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |
| C | NO-SCALE | MODRM.R/M(rw) | IMM16/IMM32(r) | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM16/IMM32(r) | N/A |
| E | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| F | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | MODRM.REG(r) | N/A |
| G | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |
| H | NO-SCALE | VVVVV(w) | MODRM.REG(r) | MODRM.R/M(r) | N/A |

### 6.59.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.59.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SBB r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| SBB r8, r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| SBB rv/mv, rv | APX-EVEX-INT | N/A | APX_F |
| SBB rv, rv/mv, rv | APX-EVEX-INT | N/A | APX_F |
| SBB r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SBB r8, r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SBB rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| SBB rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| SBB r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| SBB r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| SBB rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| SBB rv, rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| SBB rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| SBB rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |

# 6.60 SHA1MSG1

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4. D9 /r<br>SHA1MSG1 {NF=0} {ND=0} xmm1, xmm2/m128 | A | V/N.E. | APX_F<br>SHA |

## 6.60.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |

## 6.60.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.60.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SHA1MSG1 xmm1, xmm2/m128 | APX-EVEX-SHA | N/A | APX_F, SHA |

# 6.61 SHA1MSG2

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4. DA /r<br>SHA1MSG2 {NF=0} {ND=0} xmm1, xmm2/m128 | A | V/N.E. | APX_F<br>SHA |

## 6.61.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |

## 6.61.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.61.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SHA1MSG2 xmm1, xmm2/m128 | APX-EVEX-SHA | N/A | APX_F, SHA |

# 6.62 SHA1NEXTE

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4. D8 /r<br>SHA1NEXTE {NF=0} {ND=0} xmm1, xmm2/m128 | A | V/N.E. | APX_F<br>SHA |

## 6.62.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |

## 6.62.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.62.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SHA1NEXTE xmm1, xmm2/m128 | APX-EVEX-SHA | N/A | APX_F, SHA |

# 6.63 SHA1RNDS4

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4. D4 /r ib<br>SHA1RNDS4 {NF=0} {ND=0} xmm1, xmm2/m128, imm8 | A | V/N.E. | APX_F<br>SHA |

## 6.63.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | IMM8(r) | N/A |

## 6.63.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.63.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SHA1RNDS4 xmm1, xmm2/m128, imm8 | APX-EVEX-SHA | N/A | APX_F, SHA |

# 6.64 SHA256MSG1

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4. DC /r<br>SHA256MSG1 {NF=0} {ND=0} xmm1, xmm2/m128 | A | V/N.E. | APX_F<br>SHA |

## 6.64.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |

## 6.64.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.64.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SHA256MSG1 xmm1, xmm2/m128 | APX-EVEX-SHA | N/A | APX_F, SHA |

# 6.65 SHA256MSG2

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4. DD /r<br>SHA256MSG2 {NF=0} {ND=0} xmm1, xmm2/m128 | A | V/N.E. | APX_F<br>SHA |

## 6.65.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |

## 6.65.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.65.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SHA256MSG2 xmm1, xmm2/m128 | APX-EVEX-SHA | N/A | APX_F, SHA |

# 6.66 SHA256RNDS2

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4. DB /r<br>SHA256RNDS2 {NF=0} {ND=0} xmm1, xmm2/m128 | A | V/N.E. | APX_F<br>SHA |

## 6.66.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |

## 6.66.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.66.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SHA256RNDS2 xmm1, xmm2/m128, <xmm0> | APX-EVEX-SHA | N/A | APX_F, SHA |

## 6.67 SHL

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED C0 /4 ib<br>SHL {NF} {ND=0} r8/m8, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED C0 /4 ib<br>SHL {NF} {ND=1} r8, r8/m8, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED C0 /6 ib<br>SHL {NF} {ND=0} r8/m8, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED C0 /6 ib<br>SHL {NF} {ND=1} r8, r8/m8, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /4 ib<br>SHL {NF} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /4 ib<br>SHL {NF} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /4 ib<br>SHL {NF} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /4 ib<br>SHL {NF} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /6 ib<br>SHL {NF} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /6 ib<br>SHL {NF} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /6 ib<br>SHL {NF} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /6 ib<br>SHL {NF} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /4<br>SHL {NF} {ND=0} r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /4<br>SHL {NF} {ND=1} r8, r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /6<br>SHL {NF} {ND=0} r8/m8 | B | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED D0 /6<br>SHL {NF} {ND=1} r8, r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /4<br>SHL {NF} {ND=0} rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /4<br>SHL {NF} {ND=0} rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /4<br>SHL {NF} {ND=1} rv, rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /4<br>SHL {NF} {ND=1} rv, rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /6<br>SHL {NF} {ND=0} rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /6<br>SHL {NF} {ND=0} rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /6<br>SHL {NF} {ND=1} rv, rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /6<br>SHL {NF} {ND=1} rv, rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /4<br>SHL {NF} {ND=0} r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /4<br>SHL {NF} {ND=1} r8, r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /6<br>SHL {NF} {ND=0} r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /6<br>SHL {NF} {ND=1} r8, r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /4<br>SHL {NF} {ND=0} rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D3 /4<br>SHL {NF} {ND=0} rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /4<br>SHL {NF} {ND=1} rv, rv/mv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE D3 /4 SHL {NF} {ND=1} rv, rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /6 SHL {NF} {ND=0} rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D3 /6 SHL {NF} {ND=0} rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /6 SHL {NF} {ND=1} rv, rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D3 /6 SHL {NF} {ND=1} rv, rv/mv | C | V/N.E. | APX_F |

### 6.67.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| B | NO-SCALE | MODRM.R/M(rw) | N/A | N/A | N/A |
| C | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |

### 6.67.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.67.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SHL r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |

| | | | |
|---|---|---|---|
| SHL r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| SHL rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| SHL rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| SHL r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHL r8, r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHL rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHL rv, rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHL r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHL r8, r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHL rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHL rv, rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |

# 6.68 SHLD

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE 24 /r ib<br>SHLD {NF} {ND=0} rv/mv, rv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 24 /r ib<br>SHLD {NF} {ND=0} rv/mv, rv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 24 /r ib<br>SHLD {NF} {ND=1} rv, rv/mv, rv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 24 /r ib<br>SHLD {NF} {ND=1} rv, rv/mv, rv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE A5 /r<br>SHLD {NF} {ND=0} rv/mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE A5 /r<br>SHLD {NF} {ND=0} rv/mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE A5 /r<br>SHLD {NF} {ND=1} rv, rv/mv, rv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE A5 /r<br>SHLD {NF} {ND=1} rv, rv/mv, rv | D | V/N.E. | APX_F |

## 6.68.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rcw) | MODRM.REG(r) | IMM8(r) | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | MODRM.REG(r) | IMM8(r) |
| C | NO-SCALE | MODRM.R/M(rcw) | MODRM.REG(r) | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | MODRM.REG(r) | N/A |

## 6.68.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.68.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SHLD rv/mv, rv, imm8 | APX-EVEX-INT | N/A | APX_F |
| SHLD rv, rv/mv, rv, imm8 | APX-EVEX-INT | N/A | APX_F |
| SHLD rv/mv, rv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHLD rv, rv/mv, rv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |

# 6.69 SHLX

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.66.0F38.W0 F7 /r<br>SHLX {NF=0} {ND=0} r32, m32/r32, r32 | A | V/N.E. | APX_F<br>BMI2 |
| EVEX.128.66.0F38.W1 F7 /r<br>SHLX {NF=0} {ND=0} r64, m64/r64, r64 | A | V/N.E. | APX_F<br>BMI2 |

## 6.69.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | VVVVV(r) | N/A |

## 6.69.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.69.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SHLX r32, m32/r32, r32 | APX-EVEX-BMI | N/A | APX_F, BMI2 |
| SHLX r64, m64/r64, r64 | APX-EVEX-BMI | N/A | APX_F, BMI2 |

## 6.70 SHR

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED C0 /5 ib<br>SHR {NF} {ND=0} r8/m8, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED C0 /5 ib<br>SHR {NF} {ND=1} r8, r8/m8, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /5 ib<br>SHR {NF} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /5 ib<br>SHR {NF} {ND=0} rv/mv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE C1 /5 ib<br>SHR {NF} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE C1 /5 ib<br>SHR {NF} {ND=1} rv, rv/mv, imm8 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /5<br>SHR {NF} {ND=0} r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D0 /5<br>SHR {NF} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /5<br>SHR {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /5<br>SHR {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D1 /5<br>SHR {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D1 /5<br>SHR {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /5<br>SHR {NF} {ND=0} r8/m8 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED D2 /5<br>SHR {NF} {ND=1} r8, r8/m8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /5<br>SHR {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE D3 /5 SHR {NF} {ND=0} rv/mv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE D3 /5 SHR {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE D3 /5 SHR {NF} {ND=1} rv, rv/mv | B | V/N.E. | APX_F |

### 6.70.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | N/A | N/A |
| C | NO-SCALE | MODRM.R/M(rw) | N/A | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |

### 6.70.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.70.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SHR r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| SHR r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| SHR rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| SHR rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |

| | | | |
|---|---|---|---|
| SHR r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHR r8, r8/m8, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHR rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHR rv, rv/mv, <1:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHR r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHR r8, r8/m8, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHR rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHR rv, rv/mv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |

# 6.71 SHRD

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE 2C /r ib<br>SHRD {NF} {ND=0} rv/mv, rv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 2C /r ib<br>SHRD {NF} {ND=0} rv/mv, rv, imm8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 2C /r ib<br>SHRD {NF} {ND=1} rv, rv/mv, rv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 2C /r ib<br>SHRD {NF} {ND=1} rv, rv/mv, rv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE AD /r<br>SHRD {NF} {ND=0} rv/mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE AD /r<br>SHRD {NF} {ND=0} rv/mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE AD /r<br>SHRD {NF} {ND=1} rv, rv/mv, rv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE AD /r<br>SHRD {NF} {ND=1} rv, rv/mv, rv | D | V/N.E. | APX_F |

## 6.71.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rcw) | MODRM.REG(r) | IMM8(r) | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | MODRM.REG(r) | IMM8(r) |
| C | NO-SCALE | MODRM.R/M(rcw) | MODRM.REG(r) | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | MODRM.REG(r) | N/A |

## 6.71.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.71.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SHRD rv/mv, rv, imm8 | APX-EVEX-INT | N/A | APX_F |
| SHRD rv, rv/mv, rv, imm8 | APX-EVEX-INT | N/A | APX_F |
| SHRD rv/mv, rv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |
| SHRD rv, rv/mv, rv, <cl:r:impl> | APX-EVEX-INT | N/A | APX_F |

# 6.72 SHRX

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.F2.0F38.W0 F7 /r<br>SHRX {NF=0} {ND=0} r32, m32/r32, r32 | A | V/N.E. | APX_F<br>BMI2 |
| EVEX.128.F2.0F38.W1 F7 /r<br>SHRX {NF=0} {ND=0} r64, m64/r64, r64 | A | V/N.E. | APX_F<br>BMI2 |

## 6.72.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | VVVVV(r) | N/A |

## 6.72.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.72.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SHRX r32, m32/r32, r32 | APX-EVEX-BMI | N/A | APX_F, BMI2 |
| SHRX r64, m64/r64, r64 | APX-EVEX-BMI | N/A | APX_F, BMI2 |

# 6.73 STTILECFG

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.66.0F38.W0 49 !(11):000:bbb<br>STTILECFG {NF=0} {ND=0} m512 | A | V/N.E. | APX_F<br>AMX-TILE |

## 6.73.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | | MODRM.R/M(w) | N/A | N/A | N/A |

## 6.73.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.73.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| STTILECFG m512 | AMX-E2-EVEX | N/A | APX_F, AMX-TILE |

## 6.74 SUB

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED 28 /r<br>SUB {NF} {ND=0} r8/m8, r8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 28 /r<br>SUB {NF} {ND=1} r8, r8/m8, r8 | F | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 29 /r<br>SUB {NF} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 29 /r<br>SUB {NF} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 29 /r<br>SUB {NF} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 29 /r<br>SUB {NF} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 2A /r<br>SUB {NF} {ND=0} r8, r8/m8 | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 2A /r<br>SUB {NF} {ND=1} r8, r8, r8/m8 | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 2B /r<br>SUB {NF} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 2B /r<br>SUB {NF} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 2B /r<br>SUB {NF} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 2B /r<br>SUB {NF} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /5 ib<br>SUB {NF} {ND=0} r8/m8, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /5 ib<br>SUB {NF} {ND=1} r8, r8/m8, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /5 id<br>SUB {NF} {ND=0} rv/mv, imm32 | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 81 /5 iw/id<br>SUB {NF} {ND=0} rv/mv, imm16/imm32 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /5 id<br>SUB {NF} {ND=1} rv, rv/mv, imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /5 iw/id<br>SUB {NF} {ND=1} rv, rv/mv, imm16/imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /5 ib<br>SUB {NF} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /5 ib<br>SUB {NF} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /5 ib<br>SUB {NF} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /5 ib<br>SUB {NF} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |

### 6.74.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | MODRM.REG(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |
| C | NO-SCALE | MODRM.R/M(rw) | IMM16/IMM32(r) | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM16/IMM32(r) | N/A |
| E | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| F | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | MODRM.REG(r) | N/A |
| G | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |
| H | NO-SCALE | VVVVV(w) | MODRM.REG(r) | MODRM.R/M(r) | N/A |

### 6.74.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.74.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SUB r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| SUB r8, r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| SUB rv/mv, rv | APX-EVEX-INT | N/A | APX_F |
| SUB rv, rv/mv, rv | APX-EVEX-INT | N/A | APX_F |
| SUB r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SUB r8, r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SUB rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| SUB rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| SUB r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| SUB r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| SUB rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| SUB rv, rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| SUB rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| SUB rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |

# 6.75 TILELOADD

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.F2.0F38.W0 4B !(11):rrr:100<br>TILELOADD {NF=0} {ND=0} tmm1, sibmem | A | V/N.E. | APX_F<br>AMX-TILE |

## 6.75.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | N/A | N/A |

## 6.75.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.75.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| TILELOADD tmm1, sibmem | AMX-E3-EVEX | N/A | APX_F, AMX-TILE |

# 6.76 TILELOADDT1

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.66.0F38.W0 4B !(11):rrr:100<br>TILELOADDT1 {NF=0} {ND=0} tmm1, sibmem | A | V/N.E. | APX_F<br>AMX-TILE |

## 6.76.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | N/A | N/A |

## 6.76.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.76.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| TILELOADDT1 tmm1, sibmem | AMX-E3-EVEX | N/A | APX_F, AMX-TILE |

# 6.77 TILESTORED

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.128.F3.0F38.W0 4B !(11):rrr:100<br>TILESTORED {NF=0} {ND=0} sibmem, tmm1 | A | V/N.E. | APX_F<br>AMX-TILE |

## 6.77.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(w) | MODRM.REG(r) | N/A | N/A |

## 6.77.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.77.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| TILESTORED sibmem, tmm1 | AMX-E3-EVEX | N/A | APX_F, AMX-TILE |

# 6.78 TZCNT

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE F4 /r<br>TZCNT {NF} {ND=0} rv, rv/mv | A | V/N.E. | APX_F<br>BMI1 |
| EVEX.LLZ.66.MAP4.SCALABLE F4 /r<br>TZCNT {NF} {ND=0} rv, rv/mv | A | V/N.E. | APX_F<br>BMI1 |

## 6.78.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | N/A | N/A |

## 6.78.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.78.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| TZCNT rv, rv/mv | APX-EVEX-INT | N/A | APX_F, BMI1 |

# 6.79 WRSSD

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.W0 66 !(11):rrr:bbb<br>WRSSD {NF=0} {ND=0} m32, r32 | A | V/N.E. | APX_F<br>CET |

## 6.79.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(w) | MODRM.REG(r) | N/A | N/A |

## 6.79.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.79.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| WRSSD m32, r32 | APX-EVEX-CET-WRSS | N/A | APX_F, CET |

# 6.80 WRSSQ

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.W1 66 !(11):rrr:bbb<br>WRSSQ {NF=0} {ND=0} m64, r64 | A | V/N.E. | APX_F<br>CET |

## 6.80.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(w) | MODRM.REG(r) | N/A | N/A |

## 6.80.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.80.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| WRSSQ m64, r64 | APX-EVEX-CET-WRSS | N/A | APX_F, CET |

# 6.81 WRUSSD

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.W0 65 !(11):rrr:bbb<br>WRUSSD {NF=0} {ND=0} m32, r32 | A | V/N.E. | APX_F<br>CET |

## 6.81.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(w) | MODRM.REG(r) | N/A | N/A |

## 6.81.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.81.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| WRUSSD m32, r32 | APX-EVEX-CET-WRUSS | N/A | APX_F, CET |

# 6.82 WRUSSQ

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.W1 65 !(11):rrr:bbb<br>WRUSSQ {NF=0} {ND=0} m64, r64 | A | V/N.E. | APX_F<br>CET |

## 6.82.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(w) | MODRM.REG(r) | N/A | N/A |

## 6.82.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

## 6.82.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| WRUSSQ m64, r64 | APX-EVEX-CET-WRUSS | N/A | APX_F, CET |

## 6.83 XOR

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED 30 /r<br>XOR {NF} {ND=0} r8/m8, r8 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 30 /r<br>XOR {NF} {ND=1} r8, r8/m8, r8 | F | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 31 /r<br>XOR {NF} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 31 /r<br>XOR {NF} {ND=0} rv/mv, rv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 31 /r<br>XOR {NF} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 31 /r<br>XOR {NF} {ND=1} rv, rv/mv, rv | F | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 32 /r<br>XOR {NF} {ND=0} r8, r8/m8 | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 32 /r<br>XOR {NF} {ND=1} r8, r8, r8/m8 | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 33 /r<br>XOR {NF} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 33 /r<br>XOR {NF} {ND=0} rv, rv/mv | G | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 33 /r<br>XOR {NF} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 33 /r<br>XOR {NF} {ND=1} rv, rv, rv/mv | H | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /6 ib<br>XOR {NF} {ND=0} r8/m8, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /6 ib<br>XOR {NF} {ND=1} r8, r8/m8, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /6 id<br>XOR {NF} {ND=0} rv/mv, imm32 | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 81 /6 iw/id<br>XOR {NF} {ND=0} rv/mv, imm16/imm32 | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /6 id<br>XOR {NF} {ND=1} rv, rv/mv, imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /6 iw/id<br>XOR {NF} {ND=1} rv, rv/mv, imm16/imm32 | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /6 ib<br>XOR {NF} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /6 ib<br>XOR {NF} {ND=0} rv/mv, imm8 | E | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /6 ib<br>XOR {NF} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /6 ib<br>XOR {NF} {ND=1} rv, rv/mv, imm8 | B | V/N.E. | APX_F |

### 6.83.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(rw) | MODRM.REG(r) | N/A | N/A |
| B | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM8(r) | N/A |
| C | NO-SCALE | MODRM.R/M(rw) | IMM16/IMM32(r) | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | IMM16/IMM32(r) | N/A |
| E | NO-SCALE | MODRM.R/M(rw) | IMM8(r) | N/A | N/A |
| F | NO-SCALE | VVVVV(w) | MODRM.R/M(r) | MODRM.REG(r) | N/A |
| G | NO-SCALE | MODRM.REG(rw) | MODRM.R/M(r) | N/A | N/A |
| H | NO-SCALE | VVVVV(w) | MODRM.REG(r) | MODRM.R/M(r) | N/A |

### 6.83.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

This instruction's description remains substantially the same as that found in Volume 2A of the Intel® 64 and IA-32 Architectures Software Developer's Manual, except being suitably modified by NDD, ZU and/or NF functionalities as explained in Section 3.1 of this document.

### 6.83.3 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| XOR r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| XOR r8, r8/m8, r8 | APX-EVEX-INT | N/A | APX_F |
| XOR rv/mv, rv | APX-EVEX-INT | N/A | APX_F |
| XOR rv, rv/mv, rv | APX-EVEX-INT | N/A | APX_F |
| XOR r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| XOR r8, r8, r8/m8 | APX-EVEX-INT | N/A | APX_F |
| XOR rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| XOR rv, rv, rv/mv | APX-EVEX-INT | N/A | APX_F |
| XOR r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| XOR r8, r8/m8, imm8 | APX-EVEX-INT | N/A | APX_F |
| XOR rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| XOR rv, rv/mv, imm16/imm32 | APX-EVEX-INT | N/A | APX_F |
| XOR rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |
| XOR rv, rv/mv, imm8 | APX-EVEX-INT | N/A | APX_F |

# Chapter 7

# INTEL® APX NEW ISA – 64-BIT DIRECT ABSOLUTE JUMP

# 7.1 JMPABS

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| REX2,NO66,NO67,NOREP MAP0 W0 A1 target64<br>JMPABS target64 | A | V/N.E. | APX_F |

## 7.1.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | | N/A | N/A | N/A | N/A |

## 7.1.2 Description

JMPABS is a 64-bit only ISA extension, and acts as a near-direct branch with an absolute target.

The 64-bit immediate operand is treated an as absolute effective address, which is subject to canonicality checks.

JMPABS is a direct, un-conditional jump, and will be treated as such from the perspective of both Intel® Perfmon and Last Branch Record (LBR) facilities. JMPABS does have unique treatment from an Intel® Processor Trace perspective in that it is designed to emit an Intel® Processor Trace TIP packet by default (unlike other direct, un-conditional jumps).

## 7.1.3 Operation

```
tempRIP = <target64 from instruction>;
IF tempRIP is not canonical:
    THEN #GP(0);
ELSE
    RIP = tempRIP;

(No flags affected)
```

## 7.1.4 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| JMPABS target64 | APX-LEGACY-JMPABS | N/A | APX_F |

# Chapter 8

# INTEL® APX NEW ISA – NEW CONDITIONAL INSTRUCTIONS

## 8.1 CCMPSCC

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPB {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPB {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPB {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPB {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPB {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPB {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPB {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPB {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPB {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPB {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPB {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPBE {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPBE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPBE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPBE {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPBE {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPBE {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPBE {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPBE {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPBE {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPBE {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPBE {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPF {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPF {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPF {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPF {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPF {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPF {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPF {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPF {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPF {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPF {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPF {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPL {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPL {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPL {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPL {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPL {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPL {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPL {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPL {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPL {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPL {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPL {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPLE {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPLE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPLE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPLE {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPLE {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPLE {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPLE {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPLE {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPLE {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPLE {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPLE {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPNB {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPNB {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPNB {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPNB {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPNB {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPNB {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPNB {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPNB {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPNB {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPNB {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPNB {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPNBE {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPNBE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPNBE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPNBE {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPNBE {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPNBE {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPNBE {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPNBE {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPNBE {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPNBE {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPNBE {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPNL {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPNL {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPNL {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPNL {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPNL {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPNL {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPNL {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPNL {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPNL {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPNL {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPNL {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPNLE {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPNLE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPNLE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPNLE {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPNLE {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPNLE {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPNLE {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPNLE {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPNLE {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPNLE {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPNLE {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPNO {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPNO {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPNO {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPNO {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPNO {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPNO {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPNO {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPNO {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPNO {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPNO {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPNO {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPNS {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPNS {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPNS {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPNS {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPNS {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPNS {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPNS {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPNS {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPNS {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPNS {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPNS {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPNZ {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPNZ {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPNZ {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPNZ {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPNZ {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPNZ {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPNZ {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPNZ {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPNZ {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPNZ {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPNZ {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPO {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPO {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPO {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPO {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPO {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPO {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPO {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPO {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPO {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPO {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPO {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPS {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPS {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPS {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPS {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPS {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPS {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPS {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPS {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPS {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPS {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPS {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPT {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPT {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPT {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPT {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPT {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPT {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPT {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPT {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPT {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPT {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPT {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 38 /r<br>CCMPZ {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 39 /r<br>CCMPZ {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 39 /r<br>CCMPZ {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 3A /r<br>CCMPZ {ND=0} r8, r8/m8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 3B /r<br>CCMPZ {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 3B /r<br>CCMPZ {ND=0} rv, rv/mv, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 80 /7 ib<br>CCMPZ {ND=0} r8/m8, imm8, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 81 /7 id<br>CCMPZ {ND=0} rv/mv, imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 81 /7 iw/id<br>CCMPZ {ND=0} rv/mv, imm16/imm32, dfv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 83 /7 ib<br>CCMPZ {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 83 /7 ib<br>CCMPZ {ND=0} rv/mv, imm8, dfv | C | V/N.E. | APX_F |

### 8.1.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(r) | MODRM.REG(r) | N/A | N/A |
| B | NO-SCALE | MODRM.REG(r) | MODRM.R/M(r) | N/A | N/A |
| C | NO-SCALE | MODRM.R/M(r) | IMM8(r) | N/A | N/A |
| D | NO-SCALE | MODRM.R/M(r) | IMM16/IMM32(r) | N/A | N/A |

### 8.1.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

CCMPscc and CTESTscc are two new sets of instructions for conditional CMP and TEST, respectively. They are encoded by promoting all opcodes of CMP and TEST, except for those forms which have no explicit GPR or memory operands, into the EVEX space and re-interpreting the EVEX payload bits as shown in the figure titled “EVEX prefix for conditional CMP and TEST” below. Note that the V and NF bits and two of the zero bits are repurposed. The ND bit is required to be set to 0. There are no EVEX versions of CMP and TEST with EVEX.ND = 1.

| EVEX | Payload byte 1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| (0x62) | R3 | X3 | B3 | R4 | B4 | 1 | 0 | 0 |

| Payload byte 2 | | | | | | | |
|---|---|---|---|---|---|---|---|
| W | OF | SF | ZF | CF | X4 | p | p |

| Payload byte 3 | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | ND=0 | SC3 | SC2 | SC1 | SC0 |

Figure 8.1: EVEX prefix for conditional CMP and TEST

The four SC* bits form a **source condition code** SCC = EVEX.[SC3,SC2,SC1,SC0], the encoding of which is the same as that of the existing x86 condition codes (SDM volume 1 appendix B), with two exceptions:

- If SCC = 0b1010, then SCC evaluates to true regardless of the status flags value.
- If SCC = 0b1011, then SCC evaluates to false regardless of the status flags value.

Consequently, the SCC cannot test the parity flag PF. In the instruction mnemonics, the SCC appears as a suffix of the mnemonic, with T and F denoting the always true/false codes described above.

The SCC is used as a predicate for controlling the conditional execution of the CCMPscc or CTESTscc instruction:

- If SCC evaluates to true on the status flags, then the CMP or TEST is executed and it updates the status flags normally. Note that the SCC = 0b1010 exception case can be used to encode unconditional CMP or TEST as a special case of CCMP or CTEST.
- If SCC evaluates to false on the status flags, then the CMP or TEST is not executed and instead the status flags are updated using DFV (Default Flags Value) as follows:
  - OF = EVEX.OF
  - SF = EVEX.SF
  - ZF = EVEX.ZF
  - CF = EVEX.CF
  - PF = EVEX.CF
  - AF = 0

  Note that the SCC = 0b1011 exception case can be used to force any desired truth assignment to the flags [OF,SF,ZF,CF] unconditionally.

  Unlike the CMOVcc extensions discussed below, SCC evaluating to false does not suppress memory faults from a memory operand.

### 8.1.3 Operation

```
// CCMP
IF (src_flags satisfies scc):
    dst_flags = compare(src1,src2)
ELSE:
    dst_flags = flags(evex.[of,sf,zf,cf]); // DFV
```

### 8.1.4 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|

| | | | |
|---|---|---|---|
| CCMPB r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPB rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPB r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPB rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPB r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPB rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPB rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPBE r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPBE rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPBE r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPBE rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPBE r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPBE rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPBE rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPF r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPF rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPF r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPF rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPF r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPF rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPF rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPL r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPL rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |

| | | | |
|---|---|---|---|
| CCMPL r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPL rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPL r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPL rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPL rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPLE r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPLE rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPLE r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPLE rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPLE r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPLE rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPLE rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNB r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNB rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNB r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNB rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNB r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNB rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNB rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNBE r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNBE rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNBE r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNBE rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |

| | | | |
|---|---|---|---|
| CCMPNBE r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNBE rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNBE rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNL r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNL rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNL r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNL rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNL r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNL rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNL rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNLE r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNLE rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNLE r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNLE rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNLE r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNLE rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNLE rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNO r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNO rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNO r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNO rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNO r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNO rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |

| | | | |
|---|---|---|---|
| CCMPNO rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNS r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNS rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNS r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNS rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNS r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNS rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNS rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNZ r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNZ rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNZ r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNZ rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNZ r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNZ rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPNZ rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPO r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPO rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPO r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPO rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPO r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPO rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPO rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPS r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |

| | | | |
|---|---|---|---|
| CCMPS rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPS r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPS rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPS r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPS rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPS rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPT r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPT rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPT r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPT rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPT r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPT rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPT rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPZ r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPZ rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPZ r8, r8/m8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPZ rv, rv/mv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPZ r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPZ rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CCMPZ rv/mv, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |

## 8.2 CFCMOVCC

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE 42 /r CFCMOVB {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 42 /r CFCMOVB {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 42 /r CFCMOVB {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 42 /r CFCMOVB {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 42 /r CFCMOVB {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 42 /r CFCMOVB {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 42 /r CFCMOVB {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 42 /r CFCMOVB {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 46 /r CFCMOVBE {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 46 /r CFCMOVBE {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 46 /r CFCMOVBE {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 46 /r CFCMOVBE {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 46 /r CFCMOVBE {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 46 /r CFCMOVBE {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 46 /r CFCMOVBE {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 46 /r CFCMOVBE {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4C /r CFCMOVL {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4C /r CFCMOVL {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4C /r CFCMOVL {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4C /r CFCMOVL {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4C /r CFCMOVL {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4C /r CFCMOVL {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4C /r CFCMOVL {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4C /r CFCMOVL {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4E /r CFCMOVLE {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4E /r CFCMOVLE {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4E /r CFCMOVLE {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4E /r CFCMOVLE {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4E /r CFCMOVLE {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4E /r CFCMOVLE {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4E /r CFCMOVLE {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 4E /r<br>CFCMOVLE {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 43 /r<br>CFCMOVNB {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 43 /r<br>CFCMOVNB {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 43 /r<br>CFCMOVNB {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 43 /r<br>CFCMOVNB {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 43 /r<br>CFCMOVNB {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 43 /r<br>CFCMOVNB {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 43 /r<br>CFCMOVNB {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 43 /r<br>CFCMOVNB {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 47 /r<br>CFCMOVNBE {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 47 /r<br>CFCMOVNBE {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 47 /r<br>CFCMOVNBE {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 47 /r<br>CFCMOVNBE {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 47 /r<br>CFCMOVNBE {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 47 /r<br>CFCMOVNBE {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 47 /r<br>CFCMOVNBE {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 47 /r CFCMOVNBE {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4D /r CFCMOVNL {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4D /r CFCMOVNL {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4D /r CFCMOVNL {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4D /r CFCMOVNL {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4D /r CFCMOVNL {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4D /r CFCMOVNL {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4D /r CFCMOVNL {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4D /r CFCMOVNL {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4F /r CFCMOVNLE {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4F /r CFCMOVNLE {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4F /r CFCMOVNLE {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4F /r CFCMOVNLE {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4F /r CFCMOVNLE {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4F /r CFCMOVNLE {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4F /r CFCMOVNLE {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 4F /r CFCMOVNLE {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 41 /r CFCMOVNO {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 41 /r CFCMOVNO {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 41 /r CFCMOVNO {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 41 /r CFCMOVNO {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 41 /r CFCMOVNO {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 41 /r CFCMOVNO {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 41 /r CFCMOVNO {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 41 /r CFCMOVNO {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4B /r CFCMOVNP {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4B /r CFCMOVNP {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4B /r CFCMOVNP {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4B /r CFCMOVNP {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4B /r CFCMOVNP {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4B /r CFCMOVNP {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4B /r CFCMOVNP {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 4B /r CFCMOVNP {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 49 /r CFCMOVNS {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 49 /r CFCMOVNS {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 49 /r CFCMOVNS {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 49 /r CFCMOVNS {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 49 /r CFCMOVNS {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 49 /r CFCMOVNS {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 49 /r CFCMOVNS {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 49 /r CFCMOVNS {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 45 /r CFCMOVNZ {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 45 /r CFCMOVNZ {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 45 /r CFCMOVNZ {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 45 /r CFCMOVNZ {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 45 /r CFCMOVNZ {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 45 /r CFCMOVNZ {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 45 /r CFCMOVNZ {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 45 /r CFCMOVNZ {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 40 /r CFCMOVO {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 40 /r CFCMOVO {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 40 /r CFCMOVO {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 40 /r CFCMOVO {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 40 /r CFCMOVO {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 40 /r CFCMOVO {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 40 /r CFCMOVO {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 40 /r CFCMOVO {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4A /r CFCMOVP {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4A /r CFCMOVP {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4A /r CFCMOVP {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4A /r CFCMOVP {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4A /r CFCMOVP {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 4A /r CFCMOVP {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 4A /r CFCMOVP {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 4A /r CFCMOVP {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 48 /r CFCMOVS {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 48 /r CFCMOVS {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 48 /r CFCMOVS {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 48 /r CFCMOVS {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 48 /r CFCMOVS {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 48 /r CFCMOVS {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 48 /r CFCMOVS {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 48 /r CFCMOVS {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 44 /r CFCMOVZ {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 44 /r CFCMOVZ {NF=0} {ND=0} rv, rv/mv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 44 /r CFCMOVZ {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 44 /r CFCMOVZ {NF=1} {ND=0} rv, rv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 44 /r CFCMOVZ {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 44 /r CFCMOVZ {NF=1} {ND=0} mv, rv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 44 /r CFCMOVZ {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 44 /r<br>CFCMOVZ {NF=1} {ND=1} rv, rv, rv/mv | D | V/N.E. | APX_F |

### 8.2.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.REG(w) | MODRM.R/M(r) | N/A | N/A |
| B | NO-SCALE | MODRM.R/M(w) | MODRM.REG(r) | N/A | N/A |
| C | NO-SCALE | MODRM.R/M(cw) | MODRM.REG(r) | N/A | N/A |
| D | NO-SCALE | VVVVV(w) | MODRM.REG(r) | MODRM.R/M(r) | N/A |

### 8.2.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

CFCMOV is a new Conditionally Faulting ("CF") CMOVcc variant, that enables fault suppression of memory operands when the source condition is false.

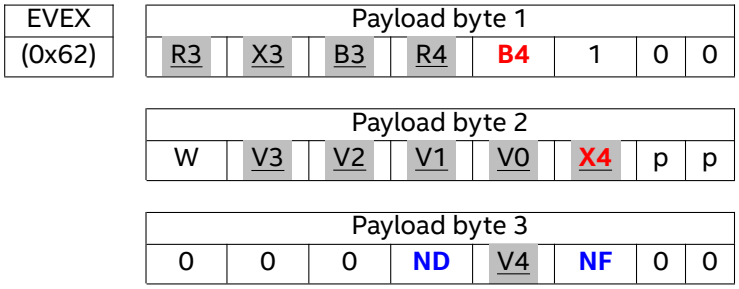


Figure 8.2: EVEX extension of CMOVcc instructions

Intel® APX introduces four different forms of EVEX-promoted CMOVcc instructions (shown in Figure 8.3) corresponding to the four possible combinations of the values of EVEX.ND and EVEX.NF (see Figure 8.2). Three of these forms have a new mnemonic, CFCMOVcc, where the "CF" prefix denotes "conditional faulting" and means that all memory faults are suppressed when the condition code evaluates to false and the r/m operand is a memory operand. Note that EVEX.NF is used as a direction bit in the 2-operand case to reverse the source and destination operands.

| EVEX.ND | EVEX.NF | Instruction Forms | Instruction Semantics |
|---|---|---|---|
| 0 | 0 | CFCMOVcc reg, r/m | IF (flags satisfies cc):<br>  reg := r/m<br>ELSE:<br>  // memory faults are suppressed<br>  reg := 0 |
| 0 | 1 | CFCMOVcc r/m, reg | IF (flags satisfies cc):<br>  r/m := reg<br>ELIF (r/m is a register):<br>  r/m := 0<br>ELSE:<br>  // memory faults are suppressed<br>  skip |
| 1 | 0 | CMOVcc ndd, reg, r/m | // memory faults are not suppressed<br>temp := r/m<br>IF (flags satisfies cc):<br>  ndd := temp<br>ELSE:<br>  ndd := reg |
| 1 | 1 | CFCMOVcc ndd, reg, r/m | IF (flags satisfies cc):<br>  ndd := r/m<br>ELSE:<br>  // memory faults are suppressed<br>  ndd := reg |

Figure 8.3: New CMOVcc variants according to EVEX.ND and EVEX.NF controls

If the destination of any of the four forms of CMOVcc and CFCMOVcc in Figure 8.3 is a register, we require that the upper bits [63:osize] of the destination register be zeroed whenever osize $<$ 64b. But if the destination is a memory location, then either osize bits are written or there is no write at all.

In contrast, the REX2 versions of CMOVcc have the same legacy behavior as the existing CMOVcc. In particular, the destination register is not zeroed and memory faults are not suppressed when the condition is false. This behavior keeps legacy CMOVcc operation semantics and timing in line with current speculation/side-channel rules used for load hardening and other usages.

### 8.2.3 Operation

```
CFCMOVcc reg, r/m (ND=0, NF=0):load

IF condition:
    temp := r/m
    reg  := temp;
ELSE:
    # Memory faults are suppressed
    # Zero dest semantics (full register write)
    reg := 0;
```

```
CFCMOVcc r/m, reg (ND=0, NF=1):store

IF condition:
    r/m := reg
ELIF (r/m is a register operand):
    # Zero dest semantics (full register write)
    r/m := 0
ELSE:
    # Memory faults are suppressed
    pass
```

```
CMOVcc ndd, reg, r/m (ND=1, NF=0):load

temp := r/m
IF condition:
    ndd := temp
ELSE:
    ndd := reg
```

```
CFMOVcc ndd, reg, r/m (ND=1, NF=1):load

IF condition:
    ndd := r/m
ELSE:
    # Memory faults are suppressed
    ndd := reg
```

### 8.2.4 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| CFCMOVB rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVB rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVB mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVB rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVBE rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVBE rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVBE mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVBE rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVL rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVL rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVL mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVL rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVLE rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVLE rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVLE mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVLE rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |

| | | | |
|---|---|---|---|
| CFCMOVNB rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNB rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNB mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNB rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNBE rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNBE rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNBE mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNBE rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNL rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNL rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNL mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNL rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNLE rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNLE rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNLE mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNLE rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNO rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNO rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNO mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNO rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNP rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNP rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNP mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |

| | | | |
|---|---|---|---|
| CFCMOVNP rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNS rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNS rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNS mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNS rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNZ rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNZ rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNZ mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVNZ rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVO rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVO rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVO mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVO rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVP rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVP rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVP mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVP rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVS rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVS rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVS mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVS rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVZ rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVZ rv, rv | APX-EVEX-CFCMOV | N/A | APX_F |

| | | | |
|---|---|---|---|
| CFCMOVZ mv, rv | APX-EVEX-CFCMOV | N/A | APX_F |
| CFCMOVZ rv, rv, rv/mv | APX-EVEX-CFCMOV | N/A | APX_F |

## 8.3 CTESTSCC

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTB {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTB {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTB {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTB {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTB {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTB {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTB {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTB {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTB {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTBE {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTBE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTBE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTBE {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTBE {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTBE {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTBE {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTBE {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTBE {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTF {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTF {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTF {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTF {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTF {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTF {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTF {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTF {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTF {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTL {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTL {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTL {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTL {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTL {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTL {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTL {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTL {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTL {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTLE {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTLE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTLE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTLE {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTLE {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTLE {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTLE {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTLE {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTLE {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTNB {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTNB {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTNB {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTNB {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTNB {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTNB {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTNB {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTNB {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTNB {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTNBE {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTNBE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTNBE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTNBE {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTNBE {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTNBE {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTNBE {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTNBE {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTNBE {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTNL {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTNL {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTNL {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTNL {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTNL {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTNL {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTNL {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTNL {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTNL {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTNLE {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTNLE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTNLE {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTNLE {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTNLE {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTNLE {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTNLE {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTNLE {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTNLE {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTNO {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTNO {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTNO {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTNO {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTNO {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTNO {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTNO {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTNO {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTNO {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTNS {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTNS {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTNS {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTNS {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTNS {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTNS {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTNS {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTNS {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTNS {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTNZ {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTNZ {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTNZ {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTNZ {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTNZ {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTNZ {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTNZ {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTNZ {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTNZ {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTO {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTO {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTO {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTO {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTO {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTO {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTO {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTO {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTO {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTS {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTS {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTS {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTS {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTS {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTS {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTS {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTS {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTS {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTT {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTT {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTT {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTT {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTT {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTT {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTT {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTT {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTT {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED 84 /r<br>CTESTZ {ND=0} r8/m8, r8, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE 85 /r<br>CTESTZ {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE 85 /r<br>CTESTZ {ND=0} rv/mv, rv, dfv | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /0 ib<br>CTESTZ {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.IGNORED F6 /1 ib<br>CTESTZ {ND=0} r8/m8, imm8, dfv | B | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /0 id<br>CTESTZ {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.66.MAP4.SCALABLE F7 /0 iw/id<br>CTESTZ {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.SCALABLE F7 /1 id<br>CTESTZ {ND=0} rv/mv, imm32, dfv | C | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.66.MAP4.SCALABLE F7 /1 iw/id<br>CTESTZ {ND=0} rv/mv, imm16/imm32, dfv | C | V/N.E. | APX_F |

### 8.3.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(r) | MODRM.REG(r) | N/A | N/A |
| B | NO-SCALE | MODRM.R/M(r) | IMM8(r) | N/A | N/A |
| C | NO-SCALE | MODRM.R/M(r) | IMM16/IMM32(r) | N/A | N/A |

### 8.3.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

CCMPscc and CTESTscc are two new sets of instructions for conditional CMP and TEST, respectively. They are encoded by promoting all opcodes of CMP and TEST, except for those forms which have no explicit GPR or memory operands, into the EVEX space and re-interpreting the EVEX payload bits as shown in the figure titled “EVEX prefix for conditional CMP and TEST” below. Note that the V and NF bits and two of the zero bits are repurposed. The ND bit is required to be set to 0. There are no EVEX versions of CMP and TEST with EVEX.ND = 1.

| EVEX | Payload byte 1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| (0x62) | R3 | X3 | B3 | R4 | B4 | 1 | 0 | 0 |

| Payload byte 2 | | | | | | | |
|---|---|---|---|---|---|---|---|
| W | OF | SF | ZF | CF | X4 | p | p |

| Payload byte 3 | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | ND=0 | SC3 | SC2 | SC1 | SC0 |

Figure 8.4: EVEX prefix for conditional CMP and TEST

The four SC* bits form a **source condition code** SCC = EVEX.[SC3,SC2,SC1,SC0], the encoding of which is the same as that of the existing x86 condition codes (SDM volume 1 appendix B), with two exceptions:

- If SCC = 0b1010, then SCC evaluates to true regardless of the status flags value.

- If SCC = 0b1011, then SCC evaluates to false regardless of the status flags value.

Consequently, the SCC cannot test the parity flag PF. In the instruction mnemonics, the SCC appears as a suffix of the mnemonic, with T and F denoting the always true/false codes described above.

The SCC is used as a predicate for controlling the conditional execution of the CCMPscc or CTESTscc instruction:

- If SCC evaluates to true on the status flags, then the CMP or TEST is executed and it updates the status flags normally. Note that the SCC = 0b1010 exception case can be used to encode unconditional CMP or TEST as a special case of CCMP or CTEST.
- If SCC evaluates to false on the status flags, then the CMP or TEST is not executed and instead the status flags are updated using DFV (Default Flags Value) as follows:
  - OF = EVEX.OF
  - SF = EVEX.SF
  - ZF = EVEX.ZF
  - CF = EVEX.CF
  - PF = EVEX.CF
  - AF = 0

  Note that the SCC = 0b1011 exception case can be used to force any desired truth assignment to the flags [OF,SF,ZF,CF] unconditionally.

  Unlike the CMOVcc extensions discussed below, SCC evaluating to false does not suppress memory faults from a memory operand.

### 8.3.3 Operation

```
// CTEST
IF (src_flags satisfies scc):
    dst_flags = test(src1,src2)
ELSE:
    dst_flags = flags(evex.[of,sf,zf,cf]);  // DFV
```

### 8.3.4 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| CTESTB r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |

| | | | |
|---|---|---|---|
| CTESTB rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTB r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTB rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTBE r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTBE rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTBE r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTBE rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTF r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTF rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTF r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTF rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTL r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTL rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTL r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTL rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTLE r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTLE rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTLE r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTLE rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNB r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNB rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNB r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNB rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |

| | | | |
|---|---|---|---|
| CTESTNBE r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNBE rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNBE r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNBE rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNL r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNL rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNL r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNL rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNLE r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNLE rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNLE r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNLE rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNO r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNO rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNO r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNO rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNS r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNS rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNS r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNS rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNZ r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNZ rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTNZ r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |

| | | | |
|---|---|---|---|
| CTESTNZ rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTO r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTO rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTO r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTO rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTS r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTS rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTS r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTS rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTT r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTT rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTT r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTT rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTZ r8/m8, r8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTZ rv/mv, rv, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTZ r8/m8, imm8, dfv | APX-EVEX-CCMP | N/A | APX_F |
| CTESTZ rv/mv, imm16/imm32, dfv | APX-EVEX-CCMP | N/A | APX_F |

## 8.4 SETCC

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F2.MAP4.IGNORED 42 /r SETB {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 46 /r SETBE {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 4C /r SETL {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 4E /r SETLE {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 43 /r SETNB {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 47 /r SETNBE {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 4D /r SETNL {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 4F /r SETNLE {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 41 /r SETNO {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 4B /r SETNP {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 49 /r SETNS {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 45 /r SETNZ {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 40 /r SETO {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 4A /r SETP {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |
| EVEX.LLZ.F2.MAP4.IGNORED 48 /r SETS {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |

*Table continued on next page...*

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.F2.MAP4.IGNORED 44 /r<br>SETZ {NF=0} {ND=ZU} r8/m8 | A | V/N.E. | APX_F |

### 8.4.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | MODRM.R/M(w) | N/A | N/A | N/A |

### 8.4.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

APX variant of SETcc, which supports zero-upper semantics (full register writer).

Sets the destination operand to 0 or 1 depending on the settings of the status flags (CF, SF, OF, ZF, and PF) in the EFLAGS register. The destination operand points to a byte register or a byte in memory. The condition code suffix (cc) indicates the condition being tested for. Additionally, if ND = 1 and the destination is a GPR, then also set the upper 48 bits of the GPR to 0.

### 8.4.3 Operation

```
IF (flags satisfies CC):
    IF (ND==1 AND dest is GPR):
        dest[63:0]=1
    ELSE:
        dest[7:0]=1
ELSE:
    IF (ND==1 AND dest is GPR):
        dest[63:0]=0
    ELSE:
        dest[7:0]=0
```

### 8.4.4 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| SETB r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETBE r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETL r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETLE r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETNB r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETNBE r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETNL r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETNLE r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETNO r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETNP r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETNS r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETNZ r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETO r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETP r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETS r8/m8 | APX-EVEX-INT | N/A | APX_F |
| SETZ r8/m8 | APX-EVEX-INT | N/A | APX_F |

# Chapter 9

# INTEL® APX NEW ISA – PUSH/POP EXTENSIONS

# 9.1 POP2

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.W0 8F 11:000:bbb<br>POP2 {NF=0} {ND=1} r64, r64 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.W1 8F 11:000:bbb<br>POP2P {NF=0} {ND=1} r64, r64 | A | V/N.E. | APX_F |

## 9.1.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | VVVVV(w) | MODRM.R/M(w) | N/A | N/A |

## 9.1.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

PUSH2 and POP2 are two new instructions for (respectively) pushing/popping 2 GPRs at a time to/from the stack.

The opcodes of PUSH2 and POP2 are those of “PUSH r/m” and “POP r/m” from legacy map 0, but we require ModRM.Mod = 3 in order to disallow memory operand. (A PUSH2 or POP2 with ModRM.Mod $\neq$ 3 triggers #UD.) In addition, we require that EVEX.ND = 1, so that the V register identifier is valid and specifies the second register operand.

The encoding and semantics of PUSH2 and POP2 are summarized in the table below, where b64 and v64 are the 64b GPRs encoded by the B and V register identifiers respectively. (The osize of PUSH2 and POP2 is always 64b.) The semantics is given in terms of an equivalent sequence of simpler instructions. We require further that neither b64 nor v64 be RSP and, for POP2, b64 and v64 be two different GPRs. Any violation of these conditions should trigger #UD. The two register values being pushed are either both written to memory or neither one is written, but the two writes are not necessarily atomic.

The data being pushed/popped by PUSH2/POP2 must be 16B-aligned on the stack. Violating this requirement should trigger #GP.

A PUSH and its corresponding POP may be marked with a 1-bit Push-Pop Acceleration (PPX) hint to indicate that the POP reads the value written by the PUSH from the stack. The processor tracks these marked instructions internally and fast-forwards register data between matching PUSH and POP

| Opcode | Instruction | Semantics |
|---|---|---|
| EVEX map=4 pp=0 ND=1 0xFF/6 Mod=3 | PUSH2 v64, b64 | PUSH v64<br>PUSH b64 |
| EVEX map=4 pp=0 ND=1 0x8F/0 Mod=3 | POP2 v64, b64 | POP v64<br>POP b64 |

Table 9.1: Encoding and semantics of PUSH2 and POP2

instructions, without going through memory or through the training loop of the Fast Store Forwarding Predictor (FSFP).

When applying the PPX hint, the compiler needs to make sure that it always marks both the PUSH and its matching POP (i.e., the POP which reads from the same stack memory address that the PUSH writes to). This balancing rule naturally applies to PUSH/POP sequences in function prologs/epilogs, respectively. It does not apply to standalone PUSH sequences, such as function argument pushes onto the stack. Such sequences should not be marked with the PPX hint.

The PPX hint is encoded by setting REX2.W = 1 and is applicable only to PUSH with opcode 0x50+rd and POP with opcode 0x58+rd in the legacy space. It is not applicable to any other variants of PUSH and POP.

The PPX hint requires the use of the REX2 prefix, even when the functional semantics can be encoded using the REX prefix or no prefix at all. Note also that the PPX hint implies OSIZE = 64b and that it is impossible to encode PPX with OSIZE = 16b, because REX2.W takes precedence over the 0x66 prefix.

Similarly, PUSH2 can be marked with a PPX hint to indicate that it has a matching POP2, which is also marked. The PPX hint for PUSH2 and POP2 is encoded by setting EVEX.W = 1. We require that EVEX.pp = 0 in PUSH2 and POP2 and their OSIZE always be 64b.

Note that for PPX to work properly, a PPX-marked PUSH2 (respectively, POP2) should always be matched with a PPX-marked POP2 (PUSH2), not with two PPX-marked POPs (PUSHs).

The PPX hint is purely a performance hint. Instructions with this hint have the same functional semantics as those without. PPX hints set by the compiler that violate the balancing rule may turn off the PPX optimization, but they will not affect program semantics.

### 9.1.3 Operation

```
// Alignment check
if (RSP % 16 != 0):
        #GP // RSP must be 16-byte aligned

// Pop 1
v64 = MEMORY[SS:RSP]
RSP = RSP + 8

// Pop 2
b64 = MEMORY[SS:RSP]
RSP = RSP + 8
```

### 9.1.4 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| POP2 r64, r64, <pop:rw:supp> | APX-EVEX-PP2 | N/A | APX_F |
| POP2P r64, r64, <pop:rw:supp> | APX-EVEX-PP2 | N/A | APX_F |

## 9.2 PUSH2

| Encoding / Instruction | Op/En | 64/32-bit mode | CPUID |
|---|---|---|---|
| EVEX.LLZ.NP.MAP4.W0 FF 11:110:bbb<br>PUSH2 {NF=0} {ND=1} r64, r64 | A | V/N.E. | APX_F |
| EVEX.LLZ.NP.MAP4.W1 FF 11:110:bbb<br>PUSH2P {NF=0} {ND=1} r64, r64 | A | V/N.E. | APX_F |

### 9.2.1 Instruction Operand Encoding

| Op/En | Tuple | Operand 1 | Operand 2 | Operand 3 | Operand 4 |
|---|---|---|---|---|---|
| A | NO-SCALE | VVVVV(r) | MODRM.R/M(r) | N/A | N/A |

### 9.2.2 Description

**Note:**

These instructions are promoted to EVEX to provide Intel® APX functionality. These instructions may have existing, inherited CPUID- and XCR0-sensitivity, independent of APX_F.

PUSH2 and POP2 are two new instructions for (respectively) pushing/popping 2 GPRs at a time to/from the stack.

The opcodes of PUSH2 and POP2 are those of “PUSH r/m” and “POP r/m” from legacy map 0, but we require ModRM.Mod = 3 in order to disallow memory operand. (A PUSH2 or POP2 with ModRM.Mod $\neq$ 3 triggers #UD.) In addition, we require that EVEX.ND = 1, so that the V register identifier is valid and specifies the second register operand.

The encoding and semantics of PUSH2 and POP2 are summarized in the table below, where b64 and v64 are the 64b GPRs encoded by the B and V register identifiers respectively. (The osize of PUSH2 and POP2 is always 64b.) The semantics is given in terms of an equivalent sequence of simpler instructions. We require further that neither b64 nor v64 be RSP and, for POP2, b64 and v64 be two different GPRs. Any violation of these conditions should trigger #UD. The two register values being pushed are either both written to memory or neither one is written, but the two writes are not necessarily atomic.

The data being pushed/popped by PUSH2/POP2 must be 16B-aligned on the stack. Violating this requirement should trigger #GP.

A PUSH and its corresponding POP may be marked with a 1-bit Push-Pop Acceleration (PPX) hint to indicate that the POP reads the value written by the PUSH from the stack. The processor tracks these marked instructions internally and fast-forwards register data between matching PUSH and POP

| Opcode | Instruction | Semantics |
|---|---|---|
| EVEX map=4 pp=0 ND=1 0xFF/6 Mod=3 | PUSH2 v64, b64 | PUSH v64<br>PUSH b64 |
| EVEX map=4 pp=0 ND=1 0x8F/0 Mod=3 | POP2 v64, b64 | POP v64<br>POP b64 |

Table 9.3: Encoding and semantics of PUSH2 and POP2

instructions, without going through memory or through the training loop of the Fast Store Forwarding Predictor (FSFP).

When applying the PPX hint, the compiler needs to make sure that it always marks both the PUSH and its matching POP (i.e., the POP which reads from the same stack memory address that the PUSH writes to). This balancing rule naturally applies to PUSH/POP sequences in function prologs/epilogs, respectively. It does not apply to standalone PUSH sequences, such as function argument pushes onto the stack. Such sequences should not be marked with the PPX hint.

The PPX hint is encoded by setting REX2.W = 1 and is applicable only to PUSH with opcode 0x50+rd and POP with opcode 0x58+rd in the legacy space. It is not applicable to any other variants of PUSH and POP.

The PPX hint requires the use of the REX2 prefix, even when the functional semantics can be encoded using the REX prefix or no prefix at all. Note also that the PPX hint implies OSIZE = 64b and that it is impossible to encode PPX with OSIZE = 16b, because REX2.W takes precedence over the 0x66 prefix.

Similarly, PUSH2 can be marked with a PPX hint to indicate that it has a matching POP2, which is also marked. The PPX hint for PUSH2 and POP2 is encoded by setting EVEX.W = 1. We require that EVEX.pp = 0 in PUSH2 and POP2 and their OSIZE always be 64b.

Note that for PPX to work properly, a PPX-marked PUSH2 (respectively, POP2) should always be matched with a PPX-marked POP2 (PUSH2), not with two PPX-marked POPs (PUSHs).

The PPX hint is purely a performance hint. Instructions with this hint have the same functional semantics as those without. PPX hints set by the compiler that violate the balancing rule may turn off the PPX optimization, but they will not affect program semantics.

### 9.2.3 Operation

```
// Alignment check
if (RSP % 16 != 0):
        #GP // RSP must be 16-byte aligned

// Push 1
RSP = RSP - 8
MEMORY[SS:RSP] = v64

// Push 2
RSP = RSP - 8
MEMORY[SS:RSP] = b64
```

### 9.2.4 Exceptions

| Instruction | Exception Type | Arithmetic Flags | CPUID |
|---|---|---|---|
| PUSH2 r64, r64, <push:rw:supp> | APX-EVEX-PP2 | N/A | APX_F |
| PUSH2P r64, r64, <push:rw:supp> | APX-EVEX-PP2 | N/A | APX_F |