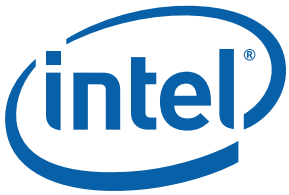

Intel® 64 and IA-32 Architectures Software Developer's Manual

Documentation Changes

March 2010

Notice: The Intel® 64 and IA-32 architectures may contain design defects or errors known as errata that may cause the product to deviate from published specifications. Current characterized errata are documented in the specification updates.

Document Number: 252046-027

Contents

Revision History

Revision	Description	Date
-001	• Initial release	November 2002
-002	• Added 1-10 Documentation Changes. • Removed old Documentation Changes items that already have been incorporated in the published Software Developer's manual	December 2002
-003	• Added 9 -17 Documentation Changes. • Removed Documentation Change #6 - References to bits Gen and Len Deleted. • Removed Documentation Change #4 - VIF Information Added to CLI Discussion	February 2003
-004	• Removed Documentation changes 1-17. • Added Documentation changes 1-24.	June 2003
-005	• Removed Documentation Changes 1-24. • Added Documentation Changes 1-15.	September 2003
-006	• Added Documentation Changes 16- 34.	November 2003
-007	• Updated Documentation changes 14, 16, 17, and 28. • Added Documentation Changes 35-45.	January 2004
-008	• Removed Documentation Changes 1-45. • Added Documentation Changes 1-5.	March 2004
-009	• Added Documentation Changes 7-27.	May 2004
-010	• Removed Documentation Changes 1-27. • Added Documentation Changes 1.	August 2004
-011	• Added Documentation Changes 2-28.	November 2004
-012	• Removed Documentation Changes 1-28. • Added Documentation Changes 1-16.	March 2005
-013	• Updated title. • There are no Documentation Changes for this revision of the document.	July 2005
-014	• Added Documentation Changes 1-21.	September 2005
-015	• Removed Documentation Changes 1-21. • Added Documentation Changes 1-20.	March 9, 2006
-016	• Added Documentation changes 21-23.	March 27, 2006
-017	• Removed Documentation Changes 1-23. • Added Documentation Changes 1-36.	September 2006
-018	• Added Documentation Changes 37-42.	October 2006
-019	• Removed Documentation Changes 1-42. • Added Documentation Changes 1-19.	March 2007
-020	• Added Documentation Changes 20-27.	May 2007
-021	• Removed Documentation Changes 1-27. • Added Documentation Changes 1-6	November 2007
-022	• Removed Documentation Changes 1-6 • Added Documentation Changes 1-6	August 2008
-023	• Removed Documentation Changes 1-6 • Added Documentation Changes 1-21	March 2009

Revision	Description	Date
-024	• Removed Documentation Changes 1-21 • Added Documentation Changes 1-16	June 2009
-025	• Removed Documentation Changes 1-16 • Added Documentation Changes 1-18	September 2009
-026	• Removed Documentation Changes 1-18 • Added Documentation Changes 1-15	December 2009
-027	• Removed Documentation Changes 1-15 • Added Documentation Changes 1-24	March 2010

§

Preface

This document is an update to the specifications contained in the Affected Documents table below. This document is a compilation of device and documentation errata, specification clarifications and changes. It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools.

Affected Documents

Document Title	Document Number/Location
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture	253665
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-M	253666
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, N-Z	253667
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1	253668
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2	253669

Nomenclature

Documentation Changes include typos, errors, or omissions from the current published specifications. These will be incorporated in any new release of the specification.

Summary Tables of Changes

The following table indicates documentation changes which apply to the Intel® 64 and IA-32 architectures. This table uses the following notations:

Codes Used in Summary Tables

Change bar to left of table row indicates this erratum is either new or modified from the previous version of the document.

Documentation Changes

No.	DOCUMENTATION CHANGES
1	Updates to Chapter 2, Volume 1
2	Updates to Chapter 12, Volume 1
3	Updates to Appendix A, Volume 1
4	Updates to Chapter 3, Volume 2A
5	Updates to Chapter 4, Volume 2B
6	Updates to Chapter 5, Volume 2B
7	Updates to Appendix A, Volume 2B
8	Updates to Appendix B, Volume 2B
9	Updates to Appendix C, Volume 2B
10	Updates to Chapter 2, Volume 3A
11	Updates to Chapter 4, Volume 3A
12	Updates to Chapter 6, Volume 3A
13	Updates to Chapter 10, Volume 3A
14	Updates to Chapter 11, Volume 3A
15	Updates to Chapter 22, Volume 3B
16	Updates to Chapter 23, Volume 3B
17	Updates to Chapter 24, Volume 3B
18	Updates to Chapter 25, Volume 3B
19	Updates to Chapter 26, Volume 3B
20	Updates to Chapter 27, Volume 3B
21	Updates to Chapter 30, Volume 3B
22	Updates to Appendix A, Volume 3B
23	Updates to Appendix B, Volume 3B
24	Updates to Appendix E, Volume 3B

Documentation Changes

1. Updates to Chapter 2, Volume 1

Change bars show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

...

2.1.15 The Intel® Core™i7 Processor Family (2008-Current)

The Intel Core i7 processor 900 series support Intel 64 architecture; they are based on Intel microarchitecture codename Nehalem using 45 nm process technology. The Intel Core i7 processor and Intel Xeon processor 5500 series include the following innovative features:

- Intel® Turbo Boost Technology converts thermal headroom into higher performance.
- Intel® HyperThreading Technology in conjunction with Quadcore to provide four cores and eight threads.
- Dedicated power control unit to reduce active and idle power consumption.
- Integrated memory controller on the processor supporting three channel of DDR3 memory.
- 8 MB inclusive Intel® Smart Cache.
- Intel® QuickPath interconnect (QPI) providing point-to-point link to chipset.
- Support for SSE4.2 and SSE4.1 instruction sets.
- Second generation Intel Virtualization Technology.

2.1.16 The Intel® Xeon® Processor 7500 Series (2010)

The Intel Xeon processor 7500 and 6500 series are based on Intel microarchitecture codename Nehalem using 45 nm process technology. They support the same features described in Section 2.1.15, plus the following innovative features:

- Up to eight cores per physical processor package.
- Up to 24 MB inclusive Intel® Smart Cache.
- Provides Intel® Scalable Memory Interconnect (Intel® SMI) channels with Intel® 7500 Scalable Memory Buffer to connect to system memory.
- Advanced RAS supporting software recoverable machine check architecture.

2.1.17 2010 Intel® Core™ Processor Family (2010)

2010 Intel Core processor family spans Intel Core i7, i5 and i3 processors. They are based on Intel microarchitecture (Westmere) using 32 nm process technology. They provide the following innovative features:

- Deliver smart performance using Intel Hyper-Threading Technology plus Intel Turbo Boost Technology.

- Enhanced Intel Smart Cache and integrated memory controller.
- Intelligent power gating.
- Repartitioned platform with on-die integration of 45nm integrated graphics.
- Support for AESNI, PCLMULQDQ, SSE4.2 and SSE4.1 instruction sets.

2.1.18 The Intel® Xeon® Processor 5600 Series (2010)

The Intel Xeon processor 5600 series are based on Intel microarchitecture (Westmere) using 32 nm process technology. They support the same features described in Section 2.1.15, plus the following innovative features:

- Up to six cores per physical processor package.
- Up to 12 MB enhanced Intel® Smart Cache.
- Support for AESNI, PCLMULQDQ, SSE4.2 and SSE4.1 instruction sets.
- Flexible Intel Virtualization Technologies across processor and I/O.

...

Table 2-2 Key Features of Most Recent Intel 64 Processors

Intel Processor	Date Intro-duced	Micro-architec-ture	Top-Bin Fre-quency at Intro-duction	Tran-sistors	Register Sizes	System Bus/ QPI Link Speed	Max. Extern. Addr. Space	On-Die Caches
...								
Intel Core i7-965 Processor Extreme Edition	2008	Intel microarchitecture codename Nehalem; Quadcore; HyperThreading Technology; Intel QPI; Intel 64 Architecture; Intel Virtualization Technology.	3.20 GHz	731 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	QPI: 6.4 GT/s; Memory: 25 GB/s	64 GB	L1: 64 KB L2: 256KB L3: 8MB
Intel Core i7-620M Processor	2010	Intel Turbo Boost Technology, Intel microarchitecture (Westmere); Dualcore; HyperThreading Technology; Intel 64 Architecture; Intel Virtualization Technology., Integrated graphics	2.66 GHz	383 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128		64 GB	L1: 64 KB L2: 256KB L3: 4MB
Intel Xeon-Processor 7560	2010	Intel Turbo Boost Technology, Intel microarchitecture codename Nehalem; Eight core; HyperThreading Technology; Intel 64 Architecture; Intel Virtualization Technology.	2.26 GHz	2.3B	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	QPI: 6.4 GT/s; Memory: 50 GB/s	16 TB	L1: 64 KB L2: 256KB L3: 24MB

...

2. Updates to Chapter 12, Volume 1

Change bars show changes to Chapter 12 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1:* Basic Architecture.

CHAPTER 12.
PROGRAMMING WITH SSE3, SSSE3, SSE4 AND AESNI

The Pentium 4 processor supporting Hyper-Threading Technology (HT Technology) introduces Streaming SIMD Extensions 3 (SSE3). The Intel Xeon processor 5100 series, Intel Core 2 processor families introduced Supplemental Streaming SIMD Extensions 3 (SSSE3). SSE4 are introduced in Intel processor generations built from 45nm process technology. This chapter describes SSE3, SSSE3, SSE4, and provides information to assist in writing application programs that use these extensions.

AESNI and PCLMLQDQ are instruction extensions targeted to accelerate high-speed block encryption and cryptographic processing. Section 12.13 covers these instructions and their relationship to the Advanced Encryption Standard (AES).

...

12.13 AESNI OVERVIEW

The AESNI extension provides six instructions to accelerate symmetric block encryption/decryption of 128-bit data blocks using the Advanced Encryption Standard (AES) specified by the NIST publication FIPS 197. Specifically, two instructions (AESENC, AESENCLAST) target the AES encryption rounds, two instructions (AESDEC, AESDECLAST) target AES decryption rounds using the Equivalent Inverse Cipher. One instruction (AESIMC) targets the Inverse MixColumn transformation primitive and one instruction (AESKEYGEN) targets generation of round keys from the cipher key for the AES encryption/decryption rounds.

AES supports encryption/decryption using cipher key lengths of 128, 192, and 256 bits by processing the data block in 10, 12, 14 rounds of predefined transformations. Figure Figure 12-5 depicts the cryptographic processing of a block of 128-bit plain text into cipher text.

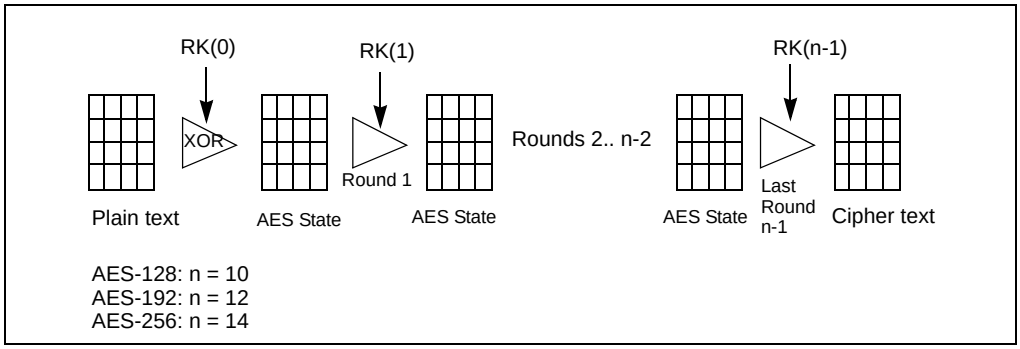

Figure 12-5 AES State Flow

The predefined AES transformation primitives are described in the next few sections, they are also referenced in the operation flow of instruction reference page of these instructions.

12.13.1 Little-Endian Architecture and Big-Endian Specification (FIPS 197)

FIPS 197 document defines the Advanced Encryption Standard (AES) and includes a set of test vectors for testing all of the steps in the algorithm, and can be used for testing and debugging.

The following observation is important for using the AES instructions offered in Intel 64 Architecture: FIPS 197 text convention is to write hex strings with the low-memory byte on the left and the high-memory byte on the right. Intel's convention is the reverse. It is similar to the difference between Big Endian and Little Endian notations.

In other words, a 128 bits vector in the FIPS document, when read from left to right, is encoded as [7:0, 15:8, 23:16, 31:24, …127:120]. Note that inside the byte, the encoding is [7:0], so the first bit from the left is the most significant bit. In practice, the test vectors are written in hexadecimal notation, where pairs of hexadecimal digits define the different bytes. To translate the FIPS 197 notation to an Intel 64 architecture compatible ("Little Endian") format, each test vector needs to be byte-reflected to [127:120,… 31:24, 23:16, 15:8, 7:0].

Example A:

FIPS Test vector: 0x000102030405060708090a0b0c0d0e0f

Intel AES Hardware: 0x0f0e0d0c0b0a09080706050403020100

It should be pointed out that the only thing at issue is a textual convention, and programmers do not need to perform byte-reversal in their code, when using the AES instructions.

12.13.1.1 AES Data Structure in Intel 64 Architecture

he AES instructions that are defined in this document operate on one or on two 128 bits source operands: State and Round Key. From the architectural point of view, the state is input in an xmm register and the Round key is input either in an xmm register or a 128-bit memory location.

In AES algorithm, the state (128 bits) can be viewed as 4 32-bit doublewords ("Word"s in AES terminology): X3, X2, X1, X0.

The state may also be viewed as a set of 16 bytes. The 16 bytes can also be viewed as a 4x4 matrix of bytes where S(i, j) with i, j = 0, 1, 2, 3 compose the 32-bit "word"s as follows:

X0 = S (3, 0) S (2, 0) S (1, 0) S (0, 0)

X1 = S (3, 1) S (2, 1) S (1, 1) S (0, 1)

X2 = S (3, 2) S (2, 2) S (1, 2) S (0, 2)

X3 = S (3, 3) S (2, 3) S (1, 3) S (0, 3)

The following tables, Table Table 12-8 through Table Table 12-11, illustrate various representations of a 128-bit state.

Table 12-8 Byte and 32-bit Word Representation of a 128-bit State

Byte #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit Position	**127 - 120**	**119 - 112**	**111 - 103**	**103 - 96**	**95 -88**	**87 -80**	**79 -72**	**71 -64**	**63 -56**	**55 -48**	**47 -40**	**39 -32**	**31 -24**	**23 -16**	**15 -8**	**7 - 0**
	127 - 96				95 - 64				64 - 32				31 - 0			
State Word	X3				X2				X1				X0			
State Byte	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

Table 12-9 Matrix Representation of a 128-bit State

A	E	I	M	S(0, 0)	S(0, 1)	S(0, 2)	S(0, 3)
B	F	J	N	S(1, 0)	S(1, 1)	S(1, 2)	S(1, 3)
C	G	K	O	S(2, 0)	S(2, 1)	S(2, 2)	S(2, 3)
D	H	L	P	S(3, 0)	S(3, 1)	S(3, 2)	S(3, 3)

Example:

FIPS vector: d4 bf 5d 30 e0 b4 52 ae b8 41 11 f1 1e 27 98 e5

This vector has the “least significant” byte d4 and the significant byte e5 (written in Big Endian format in the FIPS document). When it is translated to IA notations, the encoding is:

Table 12-10 Little Endian Representation of a 128-bit State

Byte #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
State Byte	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
State Value	e5	98	27	1e	f1	11	41	b8	ae	52	b4	e0	30	5d	bf	d4

Table 12-11 Little Endian Representation of a 4x4 Byte Matrix

A	E	I	M		d4	e0	b8	1e
B	F	J	N		bf	b4	41	27
C	G	K	O		5d	52	11	98
D	H	L	P		30	ae	f1	e5

12.13.2 AES Transformations and Functions

The following functions and transformations are used in the algorithmic descriptions of AES instruction extensions AESDEC, AESDECLAST, AESENC, AESENCLAST, AESIMC, AESKEYGENASSIST.

Note that these transformations are expressed here in a Little Endian format (and not as in the FIPS 197 document).

- MixColumns(): A byte-oriented 4x4 matrix transformation on the matrix representation of a 128-bit AES state. A FIPS-197 defined 4x4 matrix is multiplied to each 4x1

column vector of the AES state. The columns are considered polynomials with coefficients in the Finite Field that is used in the definition of FIPS 197, the operations ("multiplication" and "addition") are in that Finite Field, and the polynomials are reduced modulo x^4+1.

The MixColumns() transformation defines the relationship between each byte of the result state, represented as S'(i, j) of a 4x4 matrix (see Section 12.13.1), as a function of input state bytes, S(i, j), as follows

S'(0, j) ← FF_MUL(02H, S(0, j)) XOR FF_MUL(03H, S(1, j)) XOR S(2, j) XOR S(3, j)

S'(1, j) ← S(0, j) XOR FF_MUL(02H, S(1, j)) XOR FF_MUL(03H, S(2, j)) XOR S(3, j)

S'(2, j) ← S(0, j) XOR S(1, j) XOR FF_MUL(02H, S(2, j)) XOR FF_MUL(03H, S(3, j))

S'(3, j) ← FF_MUL(03H, S(0, j)) XOR S(1, j) XOR S(2, j) XOR FF_MUL(02H, S(3, j))

where j = 0, 1, 2, 3. FF_MUL(Byte1, Byte2) denotes the result of multiplying two elements (represented by Byte1 and byte2) in the Finite Field representation that defines AES. The result of produced bye FF_MUL(Byte1, Byte2) is an element in the Finite Field (represented as a byte). A Finite Field is a field with a finite number of elements, and when this number can be represented as a power of 2 (2n), its elements can be represented as the set of 2n binary strings of length n. AES uses a finite field with n=8 (having 256 elements). With this representation, "addition" of two elements in that field is a bit-wise XOR of their binary-string representation, producing another element in the field. Multiplication of two elements in that field is defined using an irreducible polynomial (for AES, this polynomial is $m(x) = x^8 + x^4 + x^3 + x + 1$). In this Finite Field representation, the bit value of bit position k of a byte represents the coefficient of a polynomial of order k, e.g., 1010_1101B (ADH) is represented by the polynomial $(x^7 + x^5 + x^3 + x^2 + 1)$. The byte value result of multiplication of two elements is obtained by a carry-less multiplication of the two corresponding polynomials, followed by reduction modulo the polynomial, where the remainder is calculated using operations defined in the field. For example, FF_MUL(57H, 83H) = C1H, because the carry-less polynomial multiplication of the polynomials represented by 57H and 83H produces $(x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1)$, and the remainder modulo m(x) is $(x^7 + x^6 + 1)$.

- RotWord(): performs a byte-wise cyclic permutation (rotate right in little-endian byte order) on a 32-bit AES word.

  The output word X'[j] of RotWord(X[j]) where X[j] represent the four bytes of column j, S(i, j), in descending order X[j] = (S(3, j), S(2, j), S(1, j), S(0, j)); X'[j] = (S'(3, j), S'(2, j), S'(1, j), S'(0, j)) ← (S(0, j), S(3, j), S(2, j), S(1, j))

- ShiftRows(): A byte-oriented matrix transformation that processes the matrix representation of a 16-byte AES state by cyclically shifting the last three rows of the state by different offset to the left, see Table 12-12.

Table 12-12 The ShiftRows Transformation

Matrix Representation of Input State				Output of ShiftRows			
A	E	I	M	A	E	I	M
B	F	J	N	F	J	N	B
C	G	K	O	K	O	C	G
D	H	L	P	P	D	H	L

- SubBytes(): A byte-oriented transformation that processes the 128-bit AES state by applying a non-linear substitution table (S-BOX) on each byte of the state.

  The SubBytes() function defines the relationship between each byte of the result state S'(i, j) as a function of input state byte S(i, j), by

  S'(i, j) ← S-Box (S(i, j)[7:4], S(i, j)[3:0])

  where S-BOX(S[7:4], S[3:0]) represents a look-up operation on a 16x16 table to return a byte value, see Table 12-13.

Table 12-13 Look-up Table Associated with S-Box Transformation

		S[3:0]															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S[7:4]	0	**63**	**7c**	**77**	**7b**	**f2**	**6b**	**6f**	**c5**	**30**	**01**	**67**	**2b**	**fe**	**d7**	**ab**	**76**
	1	**ca**	**82**	**c9**	**7d**	**fa**	**59**	**47**	**f0**	**ad**	**d4**	**a2**	**af**	**9c**	**a4**	**72**	**c0**
	2	**b7**	**fd**	**93**	**26**	**36**	**3f**	**f7**	**cc**	**34**	**a5**	**e5**	**f1**	**71**	**d8**	**31**	**15**
	3	**04**	**c7**	**23**	**c3**	**18**	**96**	**05**	**9a**	**07**	**12**	**80**	**e2**	**eb**	**27**	**b2**	**75**
	4	**09**	**83**	**2c**	**1a**	**1b**	**6e**	**5a**	**a0**	**52**	**3b**	**d6**	**b3**	**29**	**e3**	**2f**	**84**
	5	**53**	**d1**	**00**	**ed**	**20**	**fc**	**b1**	**5b**	**6a**	**cb**	**be**	**39**	**4a**	**4c**	**58**	**cf**
	6	**d0**	**ef**	**aa**	**fb**	**43**	**4d**	**33**	**85**	**45**	**f9**	**02**	**7f**	**50**	**3c**	**9f**	**a8**
	7	**51**	**a3**	**40**	**8f**	**92**	**9d**	**38**	**f5**	**bc**	**b6**	**da**	**21**	**10**	**ff**	**f3**	**d2**
	8	**cd**	**0c**	**13**	**ec**	**5f**	**97**	**44**	**17**	**c4**	**a7**	**7e**	**3d**	**64**	**5d**	**19**	**73**
	9	**60**	**81**	**4f**	**dc**	**22**	**2a**	**90**	**88**	**46**	**ee**	**b8**	**14**	**de**	**5e**	**0b**	**db**
	a	**e0**	**32**	**3a**	**0a**	**49**	**06**	**24**	**5c**	**c2**	**d3**	**ac**	**62**	**91**	**95**	**e4**	**79**
	b	**e7**	**c8**	**37**	**6d**	**8d**	**d5**	**4e**	**a9**	**6c**	**56**	**f4**	**ea**	**65**	**7a**	**ae**	**08**
	c	**ba**	**78**	**25**	**2e**	**1c**	**a6**	**b4**	**c6**	**e8**	**dd**	**74**	**1f**	**4b**	**bd**	**8b**	**8a**
	d	**70**	**3e**	**b5**	**66**	**48**	**03**	**f6**	**0e**	**61**	**35**	**57**	**b9**	**86**	**c1**	**1d**	**9e**
	e	**e1**	**f8**	**98**	**11**	**69**	**d9**	**8e**	**94**	**9b**	**1e**	**87**	**e9**	**ce**	**55**	**28**	**df**
	f	**8c**	**a1**	**89**	**0d**	**bf**	**e6**	**42**	**68**	**41**	**99**	**2d**	**0f**	**b0**	**54**	**bb**	**16**

- SubWord(): produces an output AES word (four bytes) from the four bytes of an input word using a non-linear substitution table (S-BOX).

  X'[j] = (S'(3, j), S'(2, j), S'(1, j), S'(0, j)) ← (S-Box (S(3, j)), S-Box(S(2, j)), S-Box(S(1, j)), S-Box(S(0, j)))

- InvMixColumns(): The inverse transformation of MixColumns().

  The InvMixColumns() transformation defines the relationship between each byte of the result state S'(i, j) as a function of input state bytes, S(i, j), by

  S'(0, j) ← FF_MUL(0eH, S(0, j)) XOR FF_MUL(0bH, S(1, j)) XOR FF_MUL(0dH, S(2, j)) XOR FF_MUL(09H, S(3, j))

  S'(1, j) ← FF_MUL(09H, S(0, j)) XOR FF_MUL(0eH, S(1, j)) XOR FF_MUL(0bH, S(2, j)) XOR FF_MUL(0dH, S(3, j))

  S'(2, j) ← FF_MUL(0dH, S(0, j)) XOR FF_MUL(09H, S(1, j)) XOR FF_MUL(0eH, S(2, j)) XOR FF_MUL(0bH, S(3, j))

S'(3, j) ← FF_MUL(0bH, S(0, j)) XOR FF_MUL(0dH, S(1, j)) XOR FF_MUL(09H, S(2, j)) XOR FF_MUL(0eH, S(3, j)), where j = 0, 1, 2, 3.

- InvShiftRows(): The inverse transformation of InvShiftRows(). The InvShiftRows() transforms the matrix representation of a 16-byte AES state by cyclically shifting the last three rows of the state by different offset to the right, see Table 12-14.

Table 12-14 The InvShiftRows Transformation

Matrix Representation of Input State				Output of ShiftRows			
A	E	I	M	A	E	I	M
B	F	J	N	N	B	F	J
C	G	K	O	K	O	C	G
D	H	L	P	H	L	P	D

- InvSubBytes(): The inverse transformation of SubBytes().

  The InvSubBytes() transformation defines the relationship between each byte of the result state S'(i, j) as a function of input state byte S(i, j), by

  S'(i, j) ← InvS-Box (S(i, j)[7:4], S(i, j)[3:0])

  where InvS-BOX(S[7:4], S[3:0]) represents a look-up operation on a 16x16 table to return a byte value, see Table 12-15.

Table 12-15 Look-up Table Associated with InvS-Box Transformation

		S[3:0]															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S[7:4]	0	**52**	**09**	**6a**	**d5**	**30**	**36**	**a5**	**38**	**bf**	**40**	**a3**	**9e**	**81**	**f3**	**d7**	**fb**
	1	**7c**	**e3**	**39**	**82**	**9b**	**2f**	**ff**	**87**	**34**	**8e**	**43**	**44**	**c4**	**de**	**e9**	**cb**
	2	**54**	**7b**	**94**	**32**	**a6**	**c2**	**23**	**3d**	**ee**	**4c**	**95**	**0b**	**42**	**fa**	**c3**	**4e**
	3	**08**	**2e**	**a1**	**66**	**28**	**d9**	**24**	**b2**	**76**	**5b**	**a2**	**49**	**6d**	**8b**	**d1**	**25**
	4	**72**	**f8**	**f6**	**64**	**86**	**68**	**98**	**16**	**d4**	**a4**	**5c**	**cc**	**5d**	**65**	**b6**	**92**
	5	**6c**	**70**	**48**	**50**	**fd**	**ed**	**b9**	**da**	**5e**	**15**	**46**	**57**	**a7**	**8d**	**9d**	**84**
	6	**90**	**d8**	**ab**	**00**	**8c**	**bc**	**d3**	**0a**	**f7**	**e4**	**58**	**05**	**b8**	**b3**	**45**	**06**
	7	**d0**	**2c**	**1e**	**8f**	**ca**	**3f**	**0f**	**02**	**c1**	**af**	**bd**	**03**	**01**	**13**	**8a**	**6b**
	8	**3a**	**91**	**11**	**41**	**4f**	**67**	**dc**	**ea**	**97**	**f2**	**cf**	**ce**	**f0**	**b4**	**e6**	**73**
	9	**96**	**ac**	**74**	**22**	**e7**	**ad**	**35**	**85**	**e2**	**f9**	**37**	**e8**	**1c**	**75**	**df**	**6e**
	a	**47**	**f1**	**1a**	**71**	**1d**	**29**	**c5**	**89**	**6f**	**b7**	**62**	**0e**	**aa**	**18**	**be**	**1b**
	b	**fc**	**56**	**3e**	**4b**	**c6**	**d2**	**79**	**20**	**9a**	**db**	**c0**	**fe**	**78**	**cd**	**5a**	**f4**
	c	**1f**	**dd**	**a8**	**33**	**88**	**07**	**c7**	**31**	**b1**	**12**	**10**	**59**	**27**	**80**	**ec**	**5f**
	d	**60**	**51**	**7f**	**a9**	**19**	**b5**	**4a**	**0d**	**2d**	**e5**	**7a**	**9f**	**93**	**c9**	**9c**	**ef**
	e	**a0**	**e0**	**3b**	**4d**	**ae**	**2a**	**f5**	**b0**	**c8**	**eb**	**bb**	**3c**	**83**	**53**	**99**	**61**
	f	**17**	**2b**	**04**	**7e**	**ba**	**77**	**d6**	**26**	**e1**	**69**	**14**	**63**	**55**	**21**	**0c**	**7d**

12.13.3 PCLMULQDQ

The PCLMULQDQ instruction performs carry-less multiplication of two 64-bit data into a 128-bit result. Carry-less multiplication of two 128-bit data into a 256-bit result can use PCLMULQDQ as building blocks.

Carry-less multiplication is a component of many cryptographic systems. It is an important piece of implementing Galois Counter Mode (GCM) operation of block ciphers. GCM operation can be used in conjunction with AES algorithms to add authentication capability. GCM usage models also include IPsec, storage standard, and security protocols over fiber channel. Additionally, PCLMULQDQ can be used in calculations of hash functions and CRC using arbitrary polynomials.

12.13.4 Checking for AESNI Support

Before an application attempts to use AESNI instructions or PCLMULQDQ, the application should follow the steps illustrated in Section 11.6.2, “Checking for SSE/SSE2 Support.” Next, use the additional step provided below:

Check that the processor supports AESNI (if CPUID.01H:ECX.AESNI[bit 25] = 1); Check that the processor supports PCLMULQDQ (if CPUID.01H:ECX.PCLMULQDQ[bit 1] = 1)

…

3. Updates to Appendix A, Volume 1

Change bars show changes to Appendix A of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1:* Basic Architecture.

…

Table A-2 EFLAGS Cross-Reference

Instruction	OF	SF	ZF	AF	PF	CF	TF	IF	DF	NT	RF
…											
BT/BTS/BTR/BTC	—	—		—	—	M					
…											

…

4. Updates to Chapter 3, Volume 2A

Change bars show changes to Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A:* Instruction Set Reference, A-M.

CHAPTER 3
INSTRUCTION SET REFERENCE, A-M

This chapter describes the instruction set for the Intel 64 and IA-32 architectures (A-M) in IA-32e, protected, Virtual-8086, and real modes of operation. The set includes general-purpose, x87 FPU, MMX, SSE/SSE2/SSE3/SSSE3/SSE4, AESNI/PCLMULQDQ, and system instructions. See also Chapter 4, "Instruction Set Reference, N-Z," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

…

AESDEC—Perform One Round of an AES Decryption Flow

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 DE /r	AESDEC xmm1, xmm2/m128	A	Valid	Valid	Perform one round of an AES decryption flow, using the Equivalent Inverse Cipher, operating on a 128-bit data (state) from xmm1 with a 128-bit round key from xmm2/m128.

Instruction Operand Encoding

Op/En	Operand 1	Operand2	Operand3	Operand4
A	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA

Description

This instruction performs a single round of the AES decryption flow using the Equivalent Inverse Cipher, with the round key from the second source operand, operating on a 128-bit data (state) from the first source operand, and store the result in the destination operand.

Use the AESDEC instruction for all but the last decryption round. For the last decryption round, use the AESDECCLAST instruction.

The first source operand and the destination operand are the same and must be an XMM register. The second source operand can be an XMM register or a 128-bit memory location.

Operation

AESDEC

STATE ← SRC1;
RoundKey ← SRC2;

```
STATE ← InvShiftRows( STATE );
STATE ← InvSubBytes( STATE );
STATE ← InvMixColumns( STATE );
DEST[127:0] ← STATE XOR RoundKey;
DEST[255:128] (Unmodified)
```

Intel C/C++ Compiler Intrinsic Equivalent

AESDEC __m128i _mm_aesdec (__m128i, __m128i)

SIMD Floating-Point Exceptions

None

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

AESDECLAST—Perform Last Round of an AES Decryption Flow

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 DF /r	AESDECLAST xmm1, xmm2/m128	A	Valid	Valid	Perform the last round of an AES decryption flow, using the Equivalent Inverse Cipher, operating on a 128-bit data (state) from xmm1 with a 128-bit round key from xmm2/m128.

Instruction Operand Encoding

Op/En	Operand 1	Operand2	Operand3	Operand4
A	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA

Description

This instruction performs the last round of the AES decryption flow using the Equivalent Inverse Cipher, with the round key from the second source operand, operating on a 128-bit data (state) from the first source operand, and store the result in the destination operand.

The first source operand and the destination operand are the same and must be an XMM register. The second source operand can be an XMM register or a 128-bit memory location.

Operation

AESDECLAST

```
STATE ← SRC1;
RoundKey ← SRC2;
STATE ← InvShiftRows( STATE );
STATE ← InvSubBytes( STATE );
DEST[127:0] ← STATE XOR RoundKey;
DEST[255:128] (Unmodified)
```

Intel C/C++ Compiler Intrinsic Equivalent

AESDECLAST __m128i _mm_aesdeclast (__m128i, __m128i)

SIMD Floating-Point Exceptions

None

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
#PF(fault-code)	For a page fault.

#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

AESENC—Perform One Round of an AES Encryption Flow

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 DC /r	AESENC xmm1, xmm2/m128	A	Valid	Valid	Perform one round of an AES encryption flow, operating on a 128-bit data (state) from xmm1 with a 128-bit round key from xmm2/m128.

Instruction Operand Encoding

Op/En	Operand 1	Operand2	Operand3	Operand4
A	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA

Description

This instruction performs a single round of an AES encryption flow using a round key from the second source operand, operating on 128-bit data (state) from the first source operand, and store the result in the destination operand.

Use the AESENC instruction for all but the last encryption rounds. For the last encryption round, use the AESENCCLAST instruction.

The first source operand and the destination operand are the same and must be an XMM register. The second source operand can be an XMM register or a 128-bit memory location.

Operation

AESENC

```
STATE ← SRC1;
RoundKey ← SRC2;
STATE ← ShiftRows( STATE );
STATE ← SubBytes( STATE );
STATE ← MixColumns( STATE );
DEST[127:0] ← STATE XOR RoundKey;
DEST[255:128] (Unmodified)
```

Intel C/C++ Compiler Intrinsic Equivalent

AESENC __m128i _mm_aesenc (__m128i, __m128i)

SIMD Floating-Point Exceptions

None

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.

If the LOCK prefix is used.

AESENCLAST—Perform Last Round of an AES Encryption Flow

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 DD /r	AESENCLAST xmm1, xmm2/m128	A	Valid	Valid	Perform the last round of an AES encryption flow, operating on a 128-bit data (state) from xmm1 with a 128-bit round key from xmm2/m128.

Instruction Operand Encoding

Op/En	Operand 1	Operand2	Operand3	Operand4
A	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA

Description

This instruction performs the last round of an AES encryption flow using a round key from the second source operand, operating on 128-bit data (state) from the first source operand, and store the result in the destination operand.

The first source operand and the destination operand are the same and must be an XMM register. The second source operand can be an XMM register or a 128-bit memory location.

Operation

AESENCLAST

STATE ← SRC1;
RoundKey ← SRC2;
STATE ← ShiftRows(STATE);
STATE ← SubBytes(STATE);
DEST[127:0] ← STATE XOR RoundKey;
DEST[255:128] (Unmodified)

Intel C/C++ Compiler Intrinsic Equivalent

AESENCLAST __m128i _mm_aesenclast (__m128i, __m128i)

SIMD Floating-Point Exceptions

None

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments. If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.

#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

AESIMC- Perform the AES InvMixColumn Transformation

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 DB /r	AESIMC xmm1, xmm2/m128	A	Valid	Valid	Perform the InvMixColumn transformation on a 128-bit round key from xmm2/m128 and store the result in xmm1.

Instruction Operand Encoding

Op/En	Operand 1	Operand2	Operand3	Operand4
A	ModRM:reg (w)	ModRM:r/m (r)	NA	NA

Description

Perform the InvMixColumns transformation on the source operand and store the result in the destination operand. The destination operand is an XMM register. The source operand can be an XMM register or a 128-bit memory location.

Note the AESIMC instruction should be applied to the expanded AES round keys (except for the first and last round key) in order to prepare them for decryption using the “Equivalent Inverse Cipher” (defined in FIPS 197).

Operation

DEST[127:0] ← InvMixColumns(SRC);
DEST[255:128] (Unmodified)

Intel C/C++ Compiler Intrinsic Equivalent

AESIMC __m128i _mm_aesimc (__m128i)

SIMD Floating-Point Exceptions

None

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

AESKEYGENASSIST - AES Round Key Generation Assist

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A DF /r ib	AESKEYGENAS-SIST xmm1, xmm2/m128, imm8	A	Valid	Valid	Assist in AES round key generation using an 8 bits Round Constant (RCON) specified in the immediate byte, operating on 128 bits of data specified in xmm2/m128 and stores the result in xmm1.

Instruction Operand Encoding

Op/En	Operand 1	Operand2	Operand3	Operand4
A	ModRM:reg (w)	ModRM:r/m (r)	imm8	NA

Description

Assist in expanding the AES cipher key, by computing steps towards generating a round key for encryption, using 128-bit data specified in the source operand and an 8-bit round constant specified as an immediate, store the result in the destination operand.

The destination operand is an XMM register. The source operand can be an XMM register or a 128-bit memory location.

Operation

```
X3[31:0] ← SRC [127: 96];
X2[31:0] ← SRC [95: 64];
X1[31:0] ← SRC [63: 32];
X0[31:0] ← SRC [31: 0];
RCON[31:0] ← ZeroExtend(Imm8[7:0]);
DEST[31:0] ← SubWord(X1);
DEST[63:32 ] ← RotWord( SubWord(X1) ) XOR RCON;
DEST[95:64] ← SubWord(X3);
DEST[127:96] ← RotWord( SubWord(X3) ) XOR RCON;
DEST[255:128] (Unmodified)
```

Intel C/C++ Compiler Intrinsic Equivalent

AESKEYGENASSIST __m128i _mm_aesimc (__m128i, const int)

SIMD Floating-Point Exceptions

None

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments. If a memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.AESNI[bit 25] = 0.
	If the LOCK prefix is used.

...

BT—Bit Test

…

Flags Affected

The CF flag contains the value of the selected bit. The ZF flag is unaffected. The OF, SF, AF, and PF flags are undefined.

…

BTC—Bit Test and Complement

…

Flags Affected

The CF flag contains the value of the selected bit before it is complemented. The ZF flag is unaffected. The OF, SF, AF, and PF flags are undefined.

…

BTR—Bit Test and Reset

…

Flags Affected

The CF flag contains the value of the selected bit before it is cleared. The ZF flag is unaffected. The OF, SF, AF, and PF flags are undefined.

…

BTS—Bit Test and Set

…

Flags Affected

The CF flag contains the value of the selected bit before it is set. The ZF flag is unaffected. The OF, SF, AF, and PF flags are undefined.

…

CLTS—Clear Task-Switched Flag in CR0

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
0F 06	CLTS	A	Valid	Valid	Clears TS flag in CR0.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
A	NA	NA	NA	NA

Description

Clears the task-switched (TS) flag in the CR0 register. This instruction is intended for use in operating-system procedures. It is a privileged instruction that can only be executed at a CPL of 0. It is allowed to be executed in real-address mode to allow initialization for protected mode.

The processor sets the TS flag every time a task switch occurs. The flag is used to synchronize the saving of FPU context in multitasking applications. See the description of the TS flag in the section titled “Control Registers” in Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*, for more information about this flag.

CLTS operation is the same in non-64-bit modes and 64-bit mode.

See Chapter 22, “VMX Non-Root Operation,” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B*, for more information about the behavior of this instruction in VMX non-root operation.

Operation

```
CR0.TS[bit 3] ← 0;
```

Flags Affected

The TS flag in CR0 register is cleared.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
#UD	If the LOCK prefix is used.

Real-Address Mode Exceptions

#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	CLTS is not recognized in virtual-8086 mode.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the CPL is greater than 0.
#UD	If the LOCK prefix is used.

...

Table 3-15. Feature Information Returned in the ECX Register

Bit #	Mnemonic	Description
0	SSE3	**Streaming SIMD Extensions 3 (SSE3)**. A value of 1 indicates the processor supports this technology.
1	PCLMULQDQ	**PCLMULQDQ**. A value of 1 indicates the processor supports the PCLMULQDQ instruction
2	DTES64	**64-bit DS Area**. A value of 1 indicates the processor supports DS area using 64-bit layout
3	MONITOR	**MONITOR/MWAIT**. A value of 1 indicates the processor supports this feature.
4	DS-CPL	**CPL Qualified Debug Store**. A value of 1 indicates the processor supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.
5	VMX	**Virtual Machine Extensions**. A value of 1 indicates that the processor supports this technology
6	SMX	**Safer Mode Extensions**. A value of 1 indicates that the processor supports this technology. See Chapter 6, "Safer Mode Extensions Reference".
7	EST	**Enhanced Intel SpeedStep® technology**. A value of 1 indicates that the processor supports this technology.
8	TM2	**Thermal Monitor 2**. A value of 1 indicates whether the processor supports this technology.
9	SSSE3	A value of 1 indicates the presence of the Supplemental Streaming SIMD Extensions 3 (SSSE3). A value of 0 indicates the instruction extensions are not present in the processor
10	CNXT-ID	**L1 Context ID**. A value of 1 indicates the L1 data cache mode can be set to either adaptive mode or shared mode. A value of 0 indicates this feature is not supported. See definition of the IA32_MISC_ENABLE MSR Bit 24 (L1 Data Cache Context Mode) for details.
12-11	Reserved	Reserved
13	CMPXCHG16B	**CMPXCHG16B Available**. A value of 1 indicates that the feature is available. See the "CMPXCHG8B/CMPXCHG16B—Compare and Exchange Bytes" section in this chapter for a description.
14	xTPR Update Control	**xTPR Update Control**. A value of 1 indicates that the processor supports changing IA32_MISC_ENABLES[bit 23].
15	PDCM	**Perfmon and Debug Capability**: A value of 1 indicates the processor supports the performance and debug feature indication MSR IA32_PERF_CAPABILITIES.
16	Reserved	Reserved
17	PCID	**Process-context identifiers**. A value of 1 indicates that the processor supports PCIDs and that software may set CR4.PCIDE to 1.

Bit #	Mnemonic	Description
18	DCA	A value of 1 indicates the processor supports the ability to prefetch data from a memory mapped device.
19	SSE4.1	A value of 1 indicates that the processor supports SSE4.1.
20	SSE4.2	A value of 1 indicates that the processor supports SSE4.2.
21	x2APIC	A value of 1 indicates that the processor supports x2APIC feature.
22	MOVBE	A value of 1 indicates that the processor supports MOVBE instruction.
23	POPCNT	A value of 1 indicates that the processor supports the POPCNT instruction.
24	Reserved	Reserved
25	AESNI	A value of 1 indicates that the processor supports the AESNI instruction extensions.
26	XSAVE	A value of 1 indicates that the processor supports the XSAVE/XRSTOR processor extended states feature, the XSETBV/XGETBV instructions, and the XFEATURE_ENABLED_MASK register (XCR0).
27	OSXSAVE	A value of 1 indicates that the OS has enabled XSETBV/XGETBV instructions to access the XFEATURE_ENABLED_MASK register (XCR0), and support for processor extended state management using XSAVE/XRSTOR.
28	AVX	A value of 1 indicates the processor supports the AVX instruction extensions.
30 - 29	Reserved	Reserved
31	Not Used	Always returns 0

…

CMOV*cc*—Conditional Move

…

Operation

```
temp ← SRC

IF condition TRUE
    THEN
        DEST ← temp;
    FI;
ELSE
    IF (OperandSize == 32 and IA-32e mode active)
        THEN
            DEST[63:32] ← 0;
    FI;
FI;
```

…

CMPS/CMPSB/CMPSW/CMPSD/CMPSQ—Compare String Operands

The CMPS, CMPSB, CMPSW, CMPSD, and CMPSQ instructions can be preceded by the REP prefix for block comparisons. More often, however, these instructions will be used in a LOOP construct that takes some action based on the setting of the status flags before the next comparison is made. See “REP/REPE/REPZ /REPNE/REPNZ—Repeat String Operation Prefix” in Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*, for a description of the REP prefix.

…

CPUID—CPU Identification

…

Table 3-12 Information Returned by CPUID Instruction

Initial EAX Value	Information Provided about the Processor	
…		
	Deterministic Cache Parameters Leaf	
04H		**NOTES:** Leaf 04H output depends on the initial value in ECX. See also: "INPUT EAX = 4: Returns Deterministic Cache Parameters for each level on page 3-214.
	EAX	Bits 4-0: Cache Type Field 0 = Null - No more caches 1 = Data Cache 2 = Instruction Cache 3 = Unified Cache 4-31 = Reserved Bits 7-5: Cache Level (starts at 1) Bits 8: Self Initializing cache level (does not need SW initialization) Bits 9: Fully Associative cache Bits 13-10: Reserved Bits 25-14: Maximum number of addressable IDs for logical processors sharing this cache*, ** Bits 31-26: Maximum number of addressable IDs for processor cores in the physical package*, ***, ****
	EBX	Bits 11-00: L = System Coherency Line Size* Bits 21-12: P = Physical Line partitions* Bits 31-22: W = Ways of associativity*
	ECX	Bits 31-00: S = Number of Sets*
	EDX	Bit 0: Write-Back Invalidate/Invalidate 0 = WBINVD/INVD from threads sharing this cache acts upon lower level caches for threads sharing this cache. 1 = WBINVD/INVD is not guaranteed to act upon lower level caches of non-originating threads sharing this cache. Bit 1: Cache Inclusiveness 0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels. Bit 2: Complex Cache Indexing 0 = Direct mapped cache. 1 = A complex function is used to index the cache, potentially using all address bits. Bits 31-03: Reserved = 0
…		

…

FXRSTOR—Restore x87 FPU, MMX , XMM, and MXCSR State

...

Description

Reloads the x87 FPU, MMX technology, XMM, and MXCSR registers from the 512-byte memory image specified in the source operand. This data should have been written to memory previously using the FXSAVE instruction, and in the same format as required by the operating modes. The first byte of the data should be located on a 16-byte boundary. There are three distinct layouts of the FXSAVE state map: one for legacy and compatibility mode, a second format for 64-bit mode FXSAVE/FXRSTOR with REX.W=0, and the third format is for 64-bit mode with FXSAVE64/FXRSTOR64. Table 3-48 shows the layout of the legacy/compatibility mode state information in memory and describes the fields in the memory image for the FXRSTOR and FXSAVE instructions. Table Table 3-51 shows the layout of the 64-bit mode state information when REX.W is set (FXSAVE64/FXRSTOR64). Table Table 3-52 shows the layout of the 64-bit mode state information when REX.W is clear (FXSAVE/FXRSTOR).

The state image referenced with an FXRSTOR instruction must have been saved using an FXSAVE instruction or be in the same format as required by Table 3-48, Table Table 3-51, or Table Table 3-52. Referencing a state image saved with an FSAVE, FNSAVE instruction or incompatible field layout will result in an incorrect state restoration.

The FXRSTOR instruction does not flush pending x87 FPU exceptions. To check and raise exceptions when loading x87 FPU state information with the FXRSTOR instruction, use an FWAIT instruction after the FXRSTOR instruction.

If the OSFXSR bit in control register CR4 is not set, the FXRSTOR instruction may not restore the states of the XMM and MXCSR registers. This behavior is implementation dependent.

If the MXCSR state contains an unmasked exception with a corresponding status flag also set, loading the register with the FXRSTOR instruction will not result in a SIMD floating-point error condition being generated. Only the next occurrence of this unmasked exception will result in the exception being generated.

Bits 16 through 32 of the MXCSR register are defined as reserved and should be set to 0. Attempting to write a 1 in any of these bits from the saved state image will result in a general protection exception (#GP) being generated.

Bytes 464:511 of an FXSAVE image are available for software use. FXRSTOR ignores the content of bytes 464:511 in an FXSAVE state image.

...

Table 3-48 Non-64-bit-Mode Layout of FXSAVE and FXRSTOR Memory Region

15 14	13 12	11 10	9 8	7 6	5	4	3 2	1 0	
Rsrvd	CS	FPU IP		FOP	Rsrvd	FTW	FSW	FCW	0
MXCSR_MASK		MXCSR		Rsrvd	DS		FPU DP		16
Reserved		ST0/MM0							32
Reserved		ST1/MM1							48
Reserved		ST2/MM2							64
Reserved		ST3/MM3							80
Reserved		ST4/MM4							96
Reserved		ST5/MM5							112
Reserved		ST6/MM6							128
Reserved		ST7/MM7							144
XMM0									160
XMM1									176
XMM2									192
XMM3									208
XMM4									224
XMM5									240
XMM6									256
XMM7									272
Reserved									288
Reserved									304
Reserved									320
Reserved									336
Reserved									352
Reserved									368
Reserved									384
Reserved									400
Reserved									416
Reserved									432
Reserved									448
Available									464
Available									480
Available									496

...

FXSAVE—Save x87 FPU, MMX Technology, and SSE State

...

Table 3-49 Field Definitions

Field	Definition
FCW	x87 FPU Control Word (16 bits). See Figure 8-6 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for the layout of the x87 FPU control word.
FSW	x87 FPU Status Word (16 bits). See Figure 8-4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for the layout of the x87 FPU status word.
Abridged FTW	x87 FPU Tag Word (8 bits). The tag information saved here is abridged, as described in the following paragraphs.
FOP	x87 FPU Opcode (16 bits). The lower 11 bits of this field contain the opcode, upper 5 bits are reserved. See Figure 8-8 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for the layout of the x87 FPU opcode field.
FPU IP	x87 FPU Instruction Pointer Offset (32 bits). The contents of this field differ depending on the current addressing mode (32-bit or 16-bit) of the processor when the FXSAVE instruction was executed: 32-bit mode — 32-bit IP offset. 16-bit mode — low 16 bits are IP offset; high 16 bits are reserved. See "x87 FPU Instruction and Operand (Data) Pointers" in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for a description of the x87 FPU instruction pointer.
CS	x87 FPU Instruction Pointer Selector (16 bits).
FPU DP	x87 FPU Instruction Operand (Data) Pointer Offset (32 bits). The contents of this field differ depending on the current addressing mode (32-bit or 16-bit) of the processor when the FXSAVE instruction was executed: 32-bit mode — 32-bit IP offset. 16-bit mode — low 16 bits are IP offset; high 16 bits are reserved.
	See "x87 FPU Instruction and Operand (Data) Pointers" in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for a description of the x87 FPU operand pointer.
DS	x87 FPU Instruction Operand (Data) Pointer Selector (16 bits).
MXCSR	MXCSR Register State (32 bits). See Figure 10-3 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for the layout of the MXCSR register. If the OSFXSR bit in control register CR4 is not set, the FXSAVE instruction may not save this register. This behavior is implementation dependent.
MXCSR_ MASK	MXCSR_MASK (32 bits). This mask can be used to adjust values written to the MXCSR register, ensuring that reserved bits are set to 0. Set the mask bits and flags in MXCSR to the mode of operation desired for SSE and SSE2 SIMD floating-point instructions. See "Guidelines for Writing to the MXCSR Register" in Chapter 11 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for instructions for how to determine and use the MXCSR_MASK value.

Field	Definition
ST0/MM0 through ST7/MM7	x87 FPU or MMX technology registers. These 80-bit fields contain the x87 FPU data registers or the MMX technology registers, depending on the state of the processor prior to the execution of the FXSAVE instruction. If the processor had been executing x87 FPU instruction prior to the FXSAVE instruction, the x87 FPU data registers are saved; if it had been executing MMX instructions (or SSE or SSE2 instructions that operated on the MMX technology registers), the MMX technology registers are saved. When the MMX technology registers are saved, the high 16 bits of the field are reserved.
XMM0 through XMM7	XMM registers (128 bits per field). If the OSFXSR bit in control register CR4 is not set, the FXSAVE instruction may not save these registers. This behavior is implementation dependent.

...

IA-32e Mode Operation

In compatibility sub-mode of IA-32e mode, legacy SSE registers, XMM0 through XMM7, are saved according to the legacy FXSAVE map. In 64-bit mode, all of the SSE registers, XMM0 through XMM15, are saved. Additionally, there are two different layouts of the FXSAVE map in 64-bit mode, corresponding to FXSAVE64 (which requires REX.W=1) and FXSAVE (REX.W=0). In the FXSAVE64 map (Table Table 3-51), the FPU IP and FPU DP pointers are 64-bit wide. In the FXSAVE map for 64-bit mode (Table Table 3-52), the FPU IP and FPU DP pointers are 32-bits.

...

Table 3-51 Layout of the 64-bit-mode FXSAVE64 Map (requires REX.W = 1)

15 14	13 12	11 10	9 8	7 6	5	4	3 2	1 0	
FPU IP				FOP	Re-served	FTW	FSW	FCW	**0**
MXCSR_MASK		MXCSR		FPU DP					**16**
Reserved		ST0/MM0							**32**
Reserved		ST1/MM1							**48**
Reserved		ST2/MM2							**64**
Reserved		ST3/MM3							**80**
Reserved		ST4/MM4							**96**
Reserved		ST5/MM5							**112**
Reserved		ST6/MM6							**128**
Reserved		ST7/MM7							**144**
XMM0									**160**
XMM1									**176**
XMM2									**192**
XMM3									**208**
XMM4									**224**
XMM5									**240**
XMM6									**256**
XMM7									**272**
XMM8									**288**
XMM9									**304**
XMM10									**320**
XMM11									**336**
XMM12									**352**
XMM13									**368**
XMM14									**384**
XMM15									**400**
Reserved									**416**
Reserved									**432**
Reserved									**448**
Available									**464**
Available									**480**
Available									**496**

…

Table 3-52 Layout of the 64-bit-mode FXSAVE Map (REX.W = 0)

15 14	13 12	11 10	9 8	7 6	5	4	3 2	1 0	
Reserved	CS	FPU IP		FOP	Re-served	FTW	FSW	FCW	**0**
MXCSR_MASK		MXCSR		Re-served			FPU DP		**16**
Reserved		ST0/MM0							**32**
Reserved		ST1/MM1							**48**
Reserved		ST2/MM2							**64**
Reserved		ST3/MM3							**80**
Reserved		ST4/MM4							**96**
Reserved		ST5/MM5							**112**
Reserved		ST6/MM6							**128**
Reserved		ST7/MM7							**144**
XMM0									**160**
XMM1									**176**
XMM2									**192**
XMM3									**208**
XMM4									**224**
XMM5									**240**
XMM6									**256**
XMM7									**272**
XMM8									**288**
XMM9									**304**
XMM10									**320**
XMM11									**336**
XMM12									**352**
XMM13									**368**
XMM14									**384**
XMM15									**400**
Reserved									**416**
Reserved									**432**
Reserved									**448**
Available									**464**
Available									**480**
Available									**496**

…

INS/INSB/INSW/INSD—Input from Port to String

…

The INS, INSB, INSW, and INSD instructions can be preceded by the REP prefix for block input of ECX bytes, words, or doublewords. See “REP/REPE/REPZ /REPNE/REPNZ—Repeat String Operation Prefix” in Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*, for a description of the REP prefix.

…

JMP—Jump

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
EB *cb*	JMP *rel8*	A	Valid	Valid	Jump short, RIP = RIP + 8-bit displacement sign extended to 64-bits
E9 *cw*	JMP *rel16*	A	N.S.	Valid	Jump near, relative, displacement relative to next instruction. Not supported in 64-bit mode.
E9 *cd*	JMP *rel32*	A	Valid	Valid	Jump near, relative, RIP = RIP + 32-bit displacement sign extended to 64-bits
FF /4	JMP *r/m16*	B	N.S.	Valid	Jump near, absolute indirect, address = zero-extended *r/m16.* Not supported in 64-bit mode.
FF /4	JMP *r/m32*	B	N.S.	Valid	Jump near, absolute indirect, address given in *r/m32.* Not supported in 64-bit mode.
FF /4	JMP *r/m64*	B	Valid	N.E.	Jump near, absolute indirect, RIP = 64-Bit offset from register or memory
EA *cd*	JMP *ptr16:16*	A	Inv.	Valid	Jump far, absolute, address given in operand
EA *cp*	JMP *ptr16:32*	A	Inv.	Valid	Jump far, absolute, address given in operand
FF /5	JMP *m16:16*	A	Valid	Valid	Jump far, absolute indirect, address given in *m16:16*
FF /5	JMP *m16:32*	A	Valid	Valid	Jump far, absolute indirect, address given in *m16:32.*
REX.W + FF /5	JMP *m16:64*	A	Valid	N.E.	Jump far, absolute indirect, address given in *m16:64.*

…

LODS/LODSB/LODSW/LODSD/LODSQ—Load String

…

The LODS, LODSB, LODSW, and LODSD instructions can be preceded by the REP prefix for block loads of ECX bytes, words, or doublewords. More often, however, these instructions are used within a LOOP construct because further processing of the data moved into the register is usually necessary before the next transfer can be made. See "REP/REPE/REPZ /REPNE/REPNZ—Repeat String Operation Prefix" in Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*, for a description of the REP prefix.

…

MASKMOVQ—Store Selected Bytes of Quadword

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F F7 /r	MASKMOVQ *mm1, mm2*	A	Valid	Valid	Selectively write bytes from *mm1* to memory location using the byte mask in *mm2*. The default memory location is specified by DS:EDI.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
A	ModRM:reg (r)	ModRM:r/m (r)	NA	NA

Description

Stores selected bytes from the source operand (first operand) into a 64-bit memory location. The mask operand (second operand) selects which bytes from the source operand are written to memory. The source and mask operands are MMX technology registers. The location of the first byte of the memory location is specified by DI/EDI and DS registers. (The size of the store address depends on the address-size attribute.)

The most significant bit in each byte of the mask operand determines whether the corresponding byte in the source operand is written to the corresponding byte location in memory: 0 indicates no write and 1 indicates write.

The MASKMOVQ instruction generates a non-temporal hint to the processor to minimize cache pollution. The non-temporal hint is implemented by using a write combining (WC) memory type protocol (see "Caching of Temporal vs. Non-Temporal Data" in Chapter 10, of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*). Because the WC protocol uses a weakly-ordered memory consistency model, a fencing operation implemented with the SFENCE or MFENCE instruction should be used in conjunction with MASKMOVQ instructions if multiple processors might use different memory types to read/write the destination memory locations.

This instruction causes a transition from x87 FPU to MMX technology state (that is, the x87 FPU top-of-stack pointer is set to 0 and the x87 FPU tag word is set to all 0s [valid]).

The behavior of the MASKMOVQ instruction with a mask of all 0s is as follows:

- No data will be written to memory.

- Transition from x87 FPU to MMX technology state will occur.
- Exceptions associated with addressing memory and page faults may still be signaled (implementation dependent).
- Signaling of breakpoints (code or data) is not guaranteed (implementation dependent).
- If the destination memory region is mapped as UC or WP, enforcement of associated semantics for these memory types is not guaranteed (that is, is reserved) and is implementation-specific.

The MASKMOVQ instruction can be used to improve performance for algorithms that need to merge data on a byte-by-byte basis. It should not cause a read for ownership; doing so generates unnecessary bandwidth since data is to be written directly using the byte-mask without allocating old data prior to the store.

In 64-bit mode, the memory address is specified by DS:RDI.

Operation

```
IF (MASK[7] = 1)
    THEN DEST[DI/EDI]  ← SRC[7:0] ELSE (* Memory location unchanged *); FI;
IF (MASK[15] = 1)
    THEN DEST[DI/EDI +1] ← SRC[15:8] ELSE (* Memory location unchanged *); FI;
    (* Repeat operation for 3rd through 6th bytes in source operand *)
IF (MASK[63] = 1)
    THEN DEST[DI/EDI +15] ← SRC[63:56] ELSE (* Memory location unchanged *); FI;
```

Intel C/C++ Compiler Intrinsic Equivalent

void _mm_maskmove_si64(__m64d, __m64n, char * p)

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments (even if mask is all 0s).
	If the destination operand is in a nonwritable segment.
	If the DS, ES, FS, or GS register contains a NULL segment selector.
#SS(0)	For an illegal address in the SS segment (even if mask is all 0s).
#PF(fault-code)	For a page fault (implementation specific).
#NM	If CR0.TS[bit 3] = 1.
#MF	If there is a pending FPU exception.
#UD	If CR0.EM[bit 2] = 1.
	If CPUID.01H:EDX.SSE[bit 25] = 0.
	If Mod field of the ModR/M byte not 11B.
	If the LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

GP	If any part of the operand lies outside the effective address space from 0 to FFFFH. (even if mask is all 0s).
#NM	If CR0.TS[bit 3] = 1.

#MF	If there is a pending FPU exception.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:EDX.SSE[bit 25] = 0.
	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

#PF(fault-code)	For a page fault (implementation specific).
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault (implementation specific).
#NM	If CR0.TS[bit 3] = 1.
#MF	If there is a pending FPU exception.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:EDX.SSE[bit 25] = 0.
	If Mod field of the ModR/M byte not 11B.
	If the LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

...

MOV—Move to/from Control Registers

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 20/*r*	MOV *r32,* CR0–CR7	A	N.E.	Valid	Move control register to *r32*
0F 20/*r*	MOV *r64,* CR0–CR7	A	Valid	N.E.	Move extended control register to *r64*.
REX.R + 0F 20 /0	MOV *r64,* CR8	A	Valid	N.E.	Move extended CR8 to *r64*.[1]
0F 22 /*r*	MOV CR0–CR7, *r32*	A	N.E.	Valid	Move *r32* to control register
0F 22 /*r*	MOV CR0–CR7, *r64*	A	Valid	N.E.	Move *r64* to extended control register.
REX.R + 0F 22 /0	MOV CR8, *r64*	A	Valid	N.E.	Move *r64* to extended CR8.[1]

NOTE:

1. MOV CR* instructions, except for MOV CR8, are serializing instructions. MOV CR8 is not architecturally defined as a serializing instruction. For more information, see Chapter 8 in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
A	ModRM:reg (w)	ModRM:r/m (r)	NA	NA

Description

Moves the contents of a control register (CR0, CR2, CR3, CR4, or CR8) to a general-purpose register or the contents of a general purpose register to a control register. The operand size for these instructions is always 32 bits in non-64-bit modes, regardless of the operand-size attribute. (See "Control Registers" in Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*, for a detailed description of the flags and fields in the control registers.) This instruction can be executed only when the current privilege level is 0.

At the opcode level, the *reg* field within the ModR/M byte specifies which of the control registers is loaded or read. The 2 bits in the *mod* field are ignored. The *r/m* field specifies the general-purpose register loaded or read. Attempts to reference CR1, CR5, CR6, CR7, and CR9–CR15 result in undefined opcode (#UD) exceptions.

When loading control registers, programs should not attempt to change the reserved bits; that is, always set reserved bits to the value previously read. An attempt to change CR4's reserved bits will cause a general protection fault. Reserved bits in CR0 and CR3 remain clear after any load of those registers; attempts to set them have no impact. On Pentium 4, Intel Xeon and P6 family processors, CR0.ET remains set after any load of CR0; attempts to clear this bit have no impact.

In certain cases, these instructions have the side effect of invalidating entries in the TLBs and the paging-structure caches. See Section 4.10.4.1, "Operations that Invalidate TLBs and Paging-Structure Caches," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* for details.

The following side effects are implementation-specific for the Pentium 4, Intel Xeon, and P6 processor family: when modifying PE or PG in register CR0, or PSE or PAE in register CR4, all TLB entries are flushed, including global entries. Software should not depend on this functionality in all Intel 64 or IA-32 processors.

In 64-bit mode, the instruction's default operation size is 64 bits. The REX.R prefix must be used to access CR8. Use of REX.B permits access to additional registers (R8-R15). Use of the REX.W prefix or 66H prefix is ignored. Use of the REX.R prefix to specify a register other than CR8 causes an invalid-opcode exception. See the summary chart at the beginning of this section for encoding data and limits.

If CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 determines whether the instruction invalidates entries in the TLBs and the paging-structure caches (see Section 4.10.4.1, "Operations that Invalidate TLBs and Paging-Structure Caches," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*). The instruction does not modify bit 63 of CR3, which is reserved and always 0.

See "Changes to Instruction Behavior in VMX Non-Root Operation" in Chapter 22 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*, for more information about the behavior of this instruction in VMX non-root operation.

Operation

```
DEST ← SRC;
```

Flags Affected

The OF, SF, ZF, AF, PF, and CF flags are undefined.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If an attempt is made to write invalid bit combinations in CR0 (such as setting the PG flag to 1 when the PE flag is set to 0, or setting the CD flag to 0 when the NW flag is set to 1).
	If an attempt is made to write a 1 to any reserved bit in CR4.
	If an attempt is made to write 1 to CR4.PCIDE.
	If any of the reserved bits are set in the page-directory pointers table (PDPT) and the loading of a control register causes the PDPT to be loaded into the processor.
#UD	If the LOCK prefix is used.
	If an attempt is made to access CR1, CR5, CR6, or CR7.

Real-Address Mode Exceptions

#GP	If an attempt is made to write a 1 to any reserved bit in CR4.
	If an attempt is made to write 1 to CR4.PCIDE.
	If an attempt is made to write invalid bit combinations in CR0 (such as setting the PG flag to 1 when the PE flag is set to 0).
#UD	If the LOCK prefix is used.
	If an attempt is made to access CR1, CR5, CR6, or CR7.

Virtual-8086 Mode Exceptions

#GP(0)	These instructions cannot be executed in virtual-8086 mode.

Compatibility Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If an attempt is made to write invalid bit combinations in CR0 (such as setting the PG flag to 1 when the PE flag is set to 0, or setting the CD flag to 0 when the NW flag is set to 1).
	If an attempt is made to change CR4.PCIDE from 0 to 1 while CR3[11:0] ≠ 000H.
	If an attempt is made to clear CR0.PG[bit 31] while CR4.PCIDE = 1.
	If an attempt is made to write a 1 to any reserved bit in CR3.
	If an attempt is made to leave IA-32e mode by clearing CR4.PAE[bit 5].
#UD	If the LOCK prefix is used.
	If an attempt is made to access CR1, CR5, CR6, or CR7.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If an attempt is made to write invalid bit combinations in CR0 (such as setting the PG flag to 1 when the PE flag is set to 0, or setting the CD flag to 0 when the NW flag is set to 1).
	If an attempt is made to change CR4.PCIDE from 0 to 1 while CR3[11:0] ≠ 000H.
	If an attempt is made to clear CR0.PG[bit 31].
	If an attempt is made to write a 1 to any reserved bit in CR4.
	If an attempt is made to write a 1 to any reserved bit in CR8.
	If an attempt is made to write a 1 to any reserved bit in CR3.
	If an attempt is made to leave IA-32e mode by clearing CR4.PAE[bit 5].
#UD	If the LOCK prefix is used.
	If an attempt is made to access CR1, CR5, CR6, or CR7.
	If the REX.R prefix is used to specify a register other than CR8.

...

MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String

...

The MOVS, MOVSB, MOVSW, and MOVSD instructions can be preceded by the REP prefix (see “REP/REPE/REPZ /REPNE/REPNZ—Repeat String Operation Prefix” in Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*, for a description of the REP prefix) for block moves of ECX bytes, words, or doublewords.

...

MWAIT—Monitor Wait

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F 01 *C9*	MWAIT	A	Valid	Valid	A hint that allow the processor to stop instruction execution and enter an implementation-dependent optimized state until occurrence of a class of events.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
A	NA	NA	NA	NA

Description

MWAIT instruction provides hints to allow the processor to enter an implementation-dependent optimized state. There are two principal targeted usages: address-range monitor and advanced power management. Both usages of MWAIT require the use of the MONITOR instruction.

A CPUID feature flag (ECX bit 3; CPUID executed EAX = 1) indicates the availability of MONITOR and MWAIT in the processor. When set, MWAIT may be executed only at privilege level 0 (use at any other privilege level results in an invalid-opcode exception). The operating system or system BIOS may disable this instruction by using the IA32_MISC_ENABLES MSR; disabling MWAIT clears the CPUID feature flag and causes execution to generate an illegal opcode exception.

This instruction's operation is the same in non-64-bit modes and 64-bit mode.

MWAIT for Address Range Monitoring

For address-range monitoring, the MWAIT instruction operates with the MONITOR instruction. The two instructions allow the definition of an address at which to wait (MONITOR) and a implementation-dependent-optimized operation to commence at the wait address (MWAIT). The execution of MWAIT is a hint to the processor that it can enter an implementation-dependent-optimized state while waiting for an event or a store operation to the address range armed by MONITOR.

ECX specifies optional extensions for the MWAIT instruction. EAX may contain hints such as the preferred optimized state the processor should enter.

For Pentium 4 processors (CPUID signature family 15 and model 3), non-zero values for EAX and ECX are reserved. Later processors defined ECX=1 as a valid extension (see below).

The following cause the processor to exit the implementation-dependent-optimized state: a store to the address range armed by the MONITOR instruction, an NMI or SMI, a debug exception, a machine check exception, the BINIT# signal, the INIT# signal, and the RESET# signal. Other implementation-dependent events may also cause the processor to exit the implementation-dependent-optimized state.

In addition, an external interrupt causes the processor to exit the implementation-dependent-optimized state if either (1) the interrupt would be delivered to software

(e.g., if HLT had been executed instead of MWAIT); or (2) ECX[0] = 1. Implementation-specific conditions may result in an interrupt causing the processor to exit the implementation-dependent-optimized state even if interrupts are masked and ECX[0] = 0.

Following exit from the implementation-dependent-optimized state, control passes to the instruction following the MWAIT instruction. A pending interrupt that is not masked (including an NMI or an SMI) may be delivered before execution of that instruction. Unlike the HLT instruction, the MWAIT instruction does not support a restart at the MWAIT instruction following the handling of an SMI.

If the preceding MONITOR instruction did not successfully arm an address range or if the MONITOR instruction has not been executed prior to executing MWAIT, then the processor will not enter the implementation-dependent-optimized state. Execution will resume at the instruction following the MWAIT.

MWAIT for Power Management

MWAIT accepts a hint and optional extension to the processor that it can enter a specified target C state while waiting for an event or a store operation to the address range armed by MONITOR. Support for MWAIT extensions for power management is indicated by CPUID.05H.ECX[0] reporting 1.

EAX and ECX will be used to communicate the additional information to the MWAIT instruction, such as the kind of optimized state the processor should enter. ECX specifies optional extensions for the MWAIT instruction. EAX may contain hints such as the preferred optimized state the processor should enter. Implementation-specific conditions may cause a processor to ignore the hint and enter a different optimized state. Future processor implementations may implement several optimized “waiting” states and will select among those states based on the hint argument.

Table 3-62 describes the meaning of ECX and EAX registers for MWAIT extensions.

Table 3-62 MWAIT Extension Register (ECX)

Bits	Description
0	Treat masked interrupts as break events (e.g., if EFLAGS.IF=0). May be set only if CPUID.01H:ECX.MONITOR[bit 3] = 1.
31: 1	Reserved

Table 3-63 MWAIT Hints Register (EAX)

Bits	Description
3 : 0	Sub C-state within a C-state, indicated by bits [7:4]
7 : 4	Target C-state* Value of 0 means C1; 1 means C2 and so on Value of 01111B means C0 Note: Target C states for MWAIT extensions are processor-specific C-states, not ACPI C-states
31: 8	Reserved

Note that if MWAIT is used to enter any of the C-states that are numerically higher than C1, a store to the address range armed by the MONITOR instruction will cause the processor to exit MWAIT only if the store was originated by other processor agents. A store from non-processor agent might not cause the processor to exit MWAIT in such cases.

For additional details of MWAIT extensions, see Chapter 14, "Power and Thermal Management," of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Operation

```
(* MWAIT takes the argument in EAX as a hint extension and is architected to take the argument in ECX as an instruction extension MWAIT EAX, ECX *)
{
WHILE ( ("Monitor Hardware is in armed state")) {
    implementation_dependent_optimized_state(EAX, ECX); }
Set the state of Monitor Hardware as triggered;
}
```

Intel C/C++ Compiler Intrinsic Equivalent

MWAIT void _mm_mwait(unsigned extensions, unsigned hints)

Example

MONITOR/MWAIT instruction pair must be coded in the same loop because execution of the MWAIT instruction will trigger the monitor hardware. It is not a proper usage to execute MONITOR once and then execute MWAIT in a loop. Setting up MONITOR without executing MWAIT has no adverse effects.

Typically the MONITOR/MWAIT pair is used in a sequence, such as:

```
EAX = Logical Address(Trigger)
ECX = 0 (*Hints *)
EDX = 0 (* Hints *)

IF ( !trigger_store_happened) {
    MONITOR EAX, ECX, EDX
    IF ( !trigger_store_happened ) {
        MWAIT EAX, ECX
    }
```

}

The above code sequence makes sure that a triggering store does not happen between the first check of the trigger and the execution of the monitor instruction. Without the second check that triggering store would go un-noticed. Typical usage of MONITOR and MWAIT would have the above code sequence within a loop.

Numeric Exceptions

None

Protected Mode Exceptions

#GP(0)	If ECX[31:1] ≠ 0.
	If ECX[0] = 1 and CPUID.05H:ECX[bit 3] = 0.
#UD	If CPUID.01H:ECX.MONITOR[bit 3] = 0.
	If current privilege level is not 0.

Real Address Mode Exceptions

#GP	If ECX[31:1] ≠ 0.
	If ECX[0] = 1 and CPUID.05H:ECX[bit 3] = 0.
#UD	If CPUID.01H:ECX.MONITOR[bit 3] = 0.

Virtual 8086 Mode Exceptions

#UD	The MWAIT instruction is not recognized in virtual-8086 mode (even if CPUID.01H:ECX.MONITOR[bit 3] = 1).

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If RCX[63:1] ≠ 0.
	If RCX[0] = 1 and CPUID.05H:ECX[bit 3] = 0.
#UD	If the current privilege level is not 0.
	If CPUID.01H:ECX.MONITOR[bit 3] = 0.

...

5. Updates to Chapter 4, Volume 2B

Change bars show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B:* Instruction Set Reference, N-Z.

CHAPTER 4
INSTRUCTION SET REFERENCE, N-Z

4.1 IMM8 CONTROL BYTE OPERATION FOR PCMPESTRI / PCMPESTRM / PCMPISTRI / PCMPISTRM

The notations introduced in this section are referenced in the reference pages of PCMPESTRI, PCMPESTRM, PCMPISTRI, PCMPISTRM. The operation of the immediate control byte is common to these four string text processing instructions of SSE4.2. This section describes the common operations.

4.1.1 General Description

The operation of PCMPESTRI, PCMPESTRM, PCMPISTRI, PCMPISTRM is defined by the combination of the respective opcode and the interpretation of an immediate control byte that is part of the instruction encoding.

The opcode controls the relationship of input bytes/words to each other (determines whether the inputs terminated strings or whether lengths are expressed explicitly) as well as the desired output (index or mask).

The Imm8 Control Byte for PCMPESTRM/PCMPESTRI/PCMPISTRM/PCMPISTRI encodes a significant amount of programmable control over the functionality of those instructions. Some functionality is unique to each instruction while some is common across some or all of the four instructions. This section describes functionality which is common across the four instructions.

The arithmetic flags (ZF, CF, SF, OF, AF, PF) are set as a result of these instructions. However, the meanings of the flags have been overloaded from their typical meanings in order to provide additional information regarding the relationships of the two inputs.

PCMPxSTRx instructions perform arithmetic comparisons between all possible pairs of bytes or words, one from each packed input source operand. The boolean results of those comparisons are then aggregated in order to produce meaningful results. The Imm8 Control Byte is used to affect the interpretation of individual input elements as well as control the arithmetic comparisons used and the specific aggregation scheme.

Specifically, the Imm8 Control Byte consists of bit fields that control the following attributes:

- **Source data format —** Byte/word data element granularity, signed or unsigned elements
- **Aggregation operation —** Encodes the mode of per-element comparison operation and the aggregation of per-element comparisons into an intermediate result
- **Polarity —** Specifies intermediate processing to be performed on the intermediate result

- **Output selection —** Specifies final operation to produce the output (depending on index or mask) from the intermediate result

4.1.2 Source Data Format

Table 4-1 Source Data Format

Imm8[1:0]	Meaning	Description
00b	Unsigned bytes	Both 128-bit sources are treated as packed, unsigned bytes.
01b	Unsigned words	Both 128-bit sources are treated as packed, unsigned words.
10b	Signed bytes	Both 128-bit sources are treated as packed, signed bytes.
11b	Signed words	Both 128-bit sources are treated as packed, signed words.

If the Imm8 Control Byte has bit[0] cleared, each source contains 16 packed bytes. If the bit is set each source contains 8 packed words. If the Imm8 Control Byte has bit[1] cleared, each input contains unsigned data. If the bit is set each source contains signed data.

4.1.3 Aggregation Operation

Table 4-2 Aggregation Operation

Imm8[3:2]	Mode	Comparison
00b	Equal any	The arithmetic comparison is "equal."
01b	Ranges	Arithmetic comparison is "greater than or equal" between even indexed bytes/words of reg and each byte/word of reg/mem. Arithmetic comparison is "less than or equal" between odd indexed bytes/words of reg and each byte/word of reg/mem. (reg/mem[m] >= reg[n] for n = even, reg/mem[m] <= reg[n] for n = odd)
10b	Equal each	The arithmetic comparison is "equal."
11b	Equal ordered	The arithmetic comparison is "equal."

All 256 (64) possible comparisons are always performed. The individual Boolean results of those comparisons are referred by "BoolRes[*Reg/Mem element index, Reg element index*]." Comparisons evaluating to "True" are represented with a 1, False with a 0 (positive logic). The initial results are then aggregated into a 16-bit (8-bit) intermediate result (IntRes1) using one of the modes described in the table below, as determined by Imm8 Control Byte bit[3:2].

See Section 4.1.6 for a description of the overrideIfDataInvalid() function used in Table Table 4-3.

Table 4-3 Aggregation Operation

Mode	Pseudocode
Equal any (find characters from a set)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For j = 0 to UpperBound, j++ For i = 0 to UpperBound, i++ IntRes1[j] OR= overrideIfDataInvalid(BoolRes[j,i])
Ranges (find characters from ranges)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For j = 0 to UpperBound, j++ For i = 0 to UpperBound, i+=2 IntRes1[j] OR= (overrideIfDataInvalid(BoolRes[j,i]) AND overrideIfDataInvalid(BoolRes[j,i+1]))
Equal each (string compare)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For i = 0 to UpperBound, i++ IntRes1[i] = overrideIfDataInvalid(BoolRes[i,i])
Equal ordered (substring search)	UpperBound = imm8[0] ? 7 :15; IntRes1 = imm8[0] ? 0xFF : 0xFFFF For j = 0 to UpperBound, j++ For i = 0 to UpperBound-j, k=j to UpperBound, k++, i++ IntRes1[j] AND= overrideIfDataInvalid(BoolRes[k,i])

4.1.4 Polarity

IntRes1 may then be further modified by performing a 1's compliment, according to the value of the Imm8 Control Byte bit[4]. Optionally, a mask may be used such that only those IntRes1 bits which correspond to "valid" reg/mem input elements are complimented (note that the definition of a valid input element is dependant on the specific opcode and is defined in each opcode's description). The result of the possible negation is referred to as IntRes2.

Table 4-4 Polarity

Imm8[5:4]	Operation	Description
00b	Positive Polarity (+)	IntRes2 = IntRes1
01b	Negative Polarity (-)	IntRes2 = -1 XOR IntRes1
10b	Masked (+)	IntRes2 = IntRes1
11b	Masked (-)	IntRes2[i] = IntRes1[i] if reg/mem[i] invalid, else = ~IntRes1[i]

4.1.5 Output Selection

Table 4-5 Ouput Selection

Imm8[6]	Operation	Description
0b	Least significant index	The index returned to ECX is of the least significant set bit in IntRes2.
1b	Most significant index	The index returned to ECX is of the most significant set bit in IntRes2.

For PCMPESTRI/PCMPISTRI, the Imm8 Control Byte bit[6] is used to determine if the index is of the least significant or most significant bit of IntRes2.

Table 4-6 Output Selection

Imm8[6]	Operation	Description
0b	Bit mask	IntRes2 is returned as the mask to the least significant bits of XMM0 with zero extension to 128 bits.
1b	Byte/word mask	IntRes2 is expanded into a byte/word mask (based on imm8[1]) and placed in XMM0. The expansion is performed by replicating each bit into all of the bits of the byte/word of the same index.

Specifically for PCMPESTRM/PCMPISTRM, the Imm8 Control Byte bit[6] is used to determine if the mask is a 16 (8) bit mask or a 128 bit byte/word mask.

4.1.6 Valid/Invalid Override of Comparisons

PCMPxSTRx instructions allow for the possibility that an end-of-string (EOS) situation may occur within the 128-bit packed data value (see the instruction descriptions below for details). Any data elements on either source that are determined to be past the EOS are considered to be invalid, and the treatment of invalid data within a comparison pair varies depending on the aggregation function being performed.

In general, the individual comparison result for each element pair BoolRes[i.j] can be forced true or false if one or more elements in the pair are invalid. See Table Table 4-7.

Table 4-7 Comparison Result for Each Element Pair BoolRes[i.j]

xmm1 byte/ word	xmm2/ m128 byte/word	Imm8[3:2] = 00b (equal any)	Imm8[3:2] = 01b (ranges)	Imm8[3:2] = 10b (equal each)	Imm8[3:2] = 11b (equal ordered)
Invalid	Invalid	Force false	Force false	Force true	Force true
Invalid	Valid	Force false	Force false	Force false	Force true
Valid	Invalid	Force false	Force false	Force false	Force false
Valid	Valid	Do not force	Do not force	Do not force	Do not force

4.1.7 Summary of Im8 Control byte

Table 4-8 Summary of Imm8 Control Byte

Imm8	Description
-------0b	128-bit sources treated as 16 packed bytes.
-------1b	128-bit sources treated as 8 packed words.
------0-b	Packed bytes/words are unsigned.
------1-b	Packed bytes/words are signed.
----00--b	Mode is equal any.
----01--b	Mode is ranges.
----10--b	Mode is equal each.
----11--b	Mode is equal ordered.
---0----b	IntRes1 is unmodified.
---1----b	IntRes1 is negated (1's compliment).
--0-----b	Negation of IntRes1 is for all 16 (8) bits.
--1-----b	Negation of IntRes1 is masked by reg/mem validity.
-0------b	Index of the least significant, set, bit is used (regardless of corresponding input element validity). IntRes2 is returned in least significant bits of XMM0.
-1------b	Index of the most significant, set, bit is used (regardless of corresponding input element validity). Each bit of IntRes2 is expanded to byte/word.
0-------b	This bit currently has no defined effect, should be 0.
1-------b	This bit currently has no defined effect, should be 0.

4.1.8 Diagram Comparison and Aggregation Process

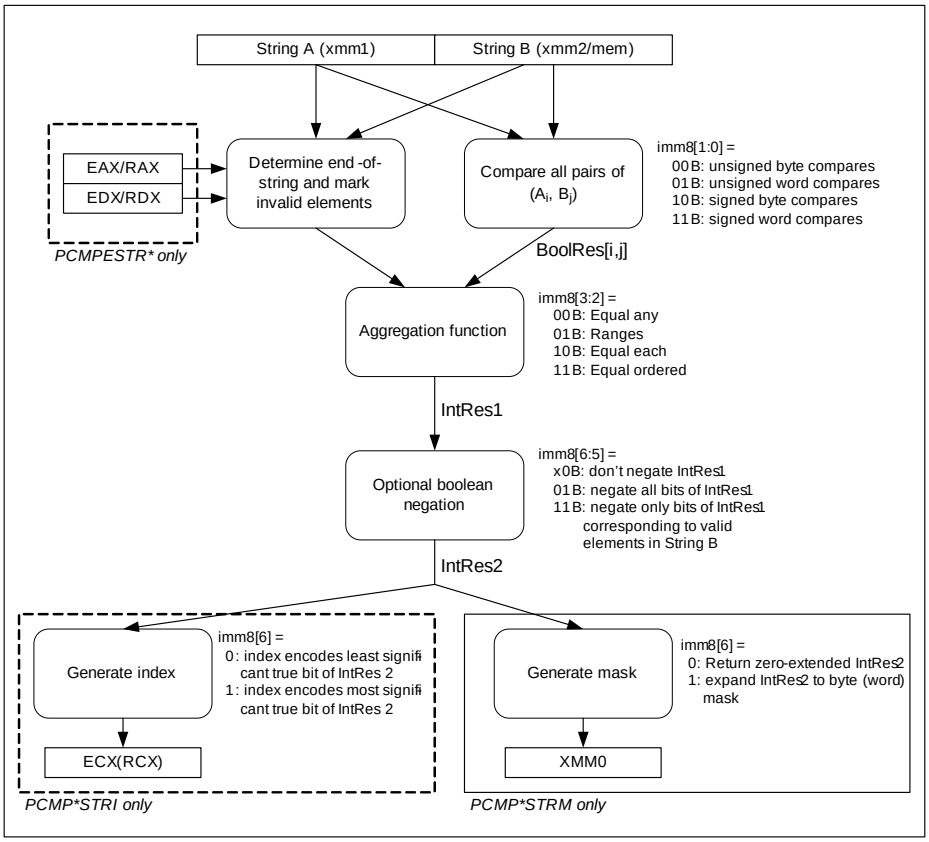

Figure 4-1 Operation of PCMPSTRx and PCMPESTRx

...

PCLMULQDQ - Carry-Less Multiplication Quadword

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 44 /r ib	PCLMULQDQ xmm1, xmm2/m128, imm8	A	Valid	Valid	Carry-less multiplication of one quadword of xmm1 by one quadword of xmm2/m128, stores the 128-bit result in xmm1. The immediate is used to determine which quadwords of xmm1 and xmm2/m128 should be used.

Instruction Operand Encoding

Op/En	Operand 1	Operand2	Operand3	Operand4
A	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA

Description

Performs a carry-less multiplication of two quadwords, selected from the first source and second source operand according to the value of the immediate byte. Bits 4 and 0 are used to select which 64-bit half of each operand to use according to Table 4-10, other bits of the immediate byte are ignored.

Table 4-10 PCLMULQDQ Quadword Selection of Immediate Byte

Imm[4]	Imm[0]	PCLMULQDQ Operation
0	0	CL_MUL(SRC2[1][63:0], SRC1[63:0])
0	1	CL_MUL(SRC2[63:0], SRC1[127:64])
1	0	CL_MUL(SRC2[127:64], SRC1[63:0])
1	1	CL_MUL(SRC2[127:64], SRC1[127:64])

NOTES:
1. SRC2 denotes the second source operand, which can be a register or memory; SRC1 denotes the first source and destination operand.

The first source operand and the destination operand are the same and must be an XMM register. The second source operand can be an XMM register or a 128-bit memory location.

Compilers and assemblers may implement the following pseudo-op syntax to simply programming and emit the required encoding for Imm8.

Table 4-11 Pseudo-Op and PCLMULQDQ Implementation

Pseudo-Op	Imm8 Encoding
PCLMULLQLQDQ *xmm1, xmm2*	0000_0000B
PCLMULHQLQDQ *xmm1, xmm2*	0000_0001B
PCLMULLQHDQ *xmm1, xmm2*	0001_0000B
PCLMULHQHDQ *xmm1, xmm2*	0001_0001B

Operation

PCLMULQDQ

```
IF (Imm8[0] = 0 )
    THEN
        TEMP1 ← SRC1 [63:0];
    ELSE
        TEMP1 ← SRC1 [127:64];
FI
IF (Imm8[4] = 0 )
    THEN
        TEMP2 ← SRC2 [63:0];
    ELSE
        TEMP2 ← SRC2 [127:64];
FI
For i = 0 to 63 {
    TmpB [ i ] ← (TEMP1[ 0 ] and TEMP2[ i ]);
    For j = 1 to i {
        TmpB [ i ] ← TmpB [ i ] xor (TEMP1[ j ] and TEMP2[ i - j ])
    }
    DEST[ i ] ← TmpB[ i ];
}
For i = 64 to 126 {
    TmpB [ i ] ← 0;
    For j = i - 63 to 63 {
        TmpB [ i ] ← TmpB [ i ] xor (TEMP1[ j ] and TEMP2[ i - j ])
    }
    DEST[ i ] ← TmpB[ i ];
}
DEST[127] ← 0;
DEST[255:128] (Unmodified)
```

Intel C/C++ Compiler Intrinsic Equivalent

PCLMULQDQ __m128i _mm_clmulepi64_si128 (__m128i, __m128i, const int)

SIMD Floating-Point Exceptions

None

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.

	If CPUID.01H:ECX.PCLMULQDQ[bit 1] = 0.
	If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.PCLMULQDQ[bit 1] = 0.
	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in real address mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.PCLMULQDQ[bit 1] = 0.
	If the LOCK prefix is used.

...

PCMPISTRI — Packed Compare Implicit Length Strings, Return Index

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
66 0F 3A 63 */r imm8*	PCMPISTRI *xmm1, xmm2/m128, imm8*	A	Valid	Valid	Perform a packed comparison of string data with implicit lengths, generating an index, and storing the result in ECX.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
A	ModRM:reg (r)	ModRM:r/m (r)	imm8	NA

Description

The instruction compares data from two strings based on the encoded value in the Imm8 Control Byte (see Section 4.1, “Imm8 Control Byte Operation for PCMPESTRI / PCMPESTRM / PCMPISTRI / PCMPISTRM”), and generates an index stored to ECX.

Each string is represented by a single value. The value is an xmm (or possibly m128 for the second operand) which contains the data elements of the string (byte or word data). Each input byte/word is augmented with a valid/invalid tag. A byte/word is considered valid only if it has a lower index than the least significant null byte/word. (The least significant null byte/word is also considered invalid.)

The comparison and aggregation operations are performed according to the encoded value of Imm8 bit fields (see Section 4.1). The index of the first (or last, according to imm8[6]) set bit of IntRes2 is returned in ECX. If no bits are set in IntRes2, ECX is set to 16 (8).

Note that the Arithmetic Flags are written in a non-standard manner in order to supply the most relevant information:

CFlag – Reset if IntRes2 is equal to zero, set otherwise
ZFlag – Set if any byte/word of xmm2/mem128 is null, reset otherwise
SFlag – Set if any byte/word of xmm1 is null, reset otherwise
OFlag –IntRes2[0]
AFlag – Reset
PFlag – Reset

Effective Operand Size

Operating mode/size	Operand1	Operand 2	Result
16 bit	xmm	xmm/m128	ECX
32 bit	xmm	xmm/m128	ECX
64 bit	xmm	xmm/m128	ECX
64 bit + REX.W	xmm	xmm/m128	RCX

Intel C/C++ Compiler Intrinsic Equivalent For Returning Index

int _mm_cmpistri (__m128i a, __m128i b, const int mode);

Intel C/C++ Compiler Intrinsics For Reading EFlag Results

int _mm_cmpistra (__m128i a, __m128i b, const int mode);
int _mm_cmpistrc (__m128i a, __m128i b, const int mode);
int _mm_cmpistro (__m128i a, __m128i b, const int mode);
int _mm_cmpistrs (__m128i a, __m128i b, const int mode);
int _mm_cmpistrz (__m128i a, __m128i b, const int mode);

SIMD Floating-Point Exceptions

N/A.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#SS(0)	For an illegal address in the SS segment.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] is 0.
	If LOCK prefix is used.
	Either the prefix REP (F3h) or REPN (F2H) is used.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] is 0.
	If LOCK prefix is used.
	Either the prefix REP (F3h) or REPN (F2H) is used.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF (fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.
	If LOCK prefix is used.
	Either the prefix REP (F3h) or REPN (F2H) is used.

PCMPISTRM — Packed Compare Implicit Length Strings, Return Mask

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
66 0F 3A 62 */r imm8*	PCMPISTRM *xmm1, xmm2/m128, imm8*	A	Valid	Valid	Perform a packed comparison of string data with implicit lengths, generating a mask, and storing the result in *XMM0.*

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
A	ModRM:reg (r)	ModRM:r/m (r)	imm8	NA

Description

The instruction compares data from two strings based on the encoded value in the imm8 byte (see Section 4.1, “Imm8 Control Byte Operation for PCMPESTRI / PCMPESTRM / PCMPISTRI / PCMPISTRM”) generating a mask stored to XMM0.

Each string is represented by a single value. The value is an xmm (or possibly m128 for the second operand) which contains the data elements of the string (byte or word data). Each input byte/word is augmented with a valid/invalid tag. A byte/word is considered valid only if it has a lower index than the least significant null byte/word. (The least significant null byte/word is also considered invalid.)

The comparison and aggregation operation are performed according to the encoded value of Imm8 bit fields (see Section 4.1). As defined by imm8[6], IntRes2 is then either stored to the least significant bits of XMM0 (zero extended to 128 bits) or expanded into a byte/word-mask and then stored to XMM0.

Note that the Arithmetic Flags are written in a non-standard manner in order to supply the most relevant information:

CFlag – Reset if IntRes2 is equal to zero, set otherwise
ZFlag – Set if any byte/word of xmm2/mem128 is null, reset otherwise
SFlag – Set if any byte/word of xmm1 is null, reset otherwise
OFlag – IntRes2[0]
AFlag – Reset
PFlag – Reset

Effective Operand Size

Operating mode/size	Operand1	Operand 2	Result
16 bit	xmm	xmm/m128	XMM0
32 bit	xmm	xmm/m128	XMM0
64 bit	xmm	xmm/m128	XMM0
64 bit + REX.W	xmm	xmm/m128	XMM0

Intel C/C++ Compiler Intrinsic Equivalent For Returning Mask

__m128i _mm_cmpistrm (__m128i a, __m128i b, const int mode);

Intel C/C++ Compiler Intrinsics For Reading EFlag Results

int _mm_cmpistra (__m128i a, __m128i b, const int mode);
int _mm_cmpistrc (__m128i a, __m128i b, const int mode);
int _mm_cmpistro (__m128i a, __m128i b, const int mode);
int _mm_cmpistrs (__m128i a, __m128i b, const int mode);
int _mm_cmpistrz (__m128i a, __m128i b, const int mode);

SIMD Floating-Point Exceptions

N/A.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#SS(0)	For an illegal address in the SS segment
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] is 0.
	If LOCK prefix is used.
	Either the prefix REP (F3h) or REPN (F2H) is used.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] is 0.
	If LOCK prefix is used.
	Either the prefix REP (F3h) or REPN (F2H) is used.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF (fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set. If OSFXSR in CR4 is 0. If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0. If LOCK prefix is used. Either the prefix REP (F3h) or REPN (F2H) is used.

...

PUSH—Push Word, Doubleword or Quadword Onto the Stack

Opcode*	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
FF /6	PUSH *r/m16*	A	Valid	Valid	Push *r/m16.*
FF /6	PUSH *r/m32*	A	N.E.	Valid	Push *r/m32.*
FF /6	PUSH *r/m64*	A	Valid	N.E.	Push *r/m64.* Default operand size 64-bits.
50+*rw*	PUSH *r16*	B	Valid	Valid	Push *r16.*
50+*rd*	PUSH *r32*	B	N.E.	Valid	Push *r32.*
50+*rd*	PUSH *r64*	B	Valid	N.E.	Push *r64.* Default operand size 64-bits.
6A	PUSH *imm8*	C	Valid	Valid	Push sign-extended *imm8. Stack pointer is decremented by the size of stack pointer.*
68	PUSH *imm16*	C	Valid	Valid	Push sign-extended *imm16. Stack pointer is decremented by the size of stack pointer.*
68	PUSH *imm32*	C	Valid	Valid	Push sign-extended *imm32. Stack pointer is decremented by the size of stack pointer.*

0E	PUSH CS	D	Invalid	Valid	Push CS.
16	PUSH SS	D	Invalid	Valid	Push SS.
1E	PUSH DS	D	Invalid	Valid	Push DS.
06	PUSH ES	D	Invalid	Valid	Push ES.
0F A0	PUSH FS	D	Valid	Valid	Push FS and decrement stack pointer by 16 bits.
0F A0	PUSH FS	D	N.E.	Valid	Push FS and decrement stack pointer by 32 bits.
0F A0	PUSH FS	D	Valid	N.E.	Push FS. Default operand size 64-bits. (66H override causes 16-bit operation).
0F A8	PUSH GS	D	Valid	Valid	Push GS and decrement stack pointer by 16 bits.
0F A8	PUSH GS	D	N.E.	Valid	Push GS and decrement stack pointer by 32 bits.
0F A8	PUSH GS	D	Valid	N.E.	Push GS, default operand size 64-bits. (66H override causes 16-bit operation).

NOTES:

* See IA-32 Architecture Compatibility section below.

…

RET—Return from Procedure

…

Operation

```
…
(* Real-address mode or virtual-8086 mode *)
IF ((PE = 0) or (PE = 1 AND VM = 1)) and instruction = far return
    THEN
        IF OperandSize = 32
            THEN
                IF top 8 bytes of stack not within stack limits
                    THEN #SS(0); FI;
                EIP ← Pop();
                CS ← Pop(); (* 32-bit pop, high-order 16 bits discarded *)
            ELSE (* OperandSize = 16 *)
                IF top 4 bytes of stack not within stack limits
                    THEN #SS(0); FI;
                tempEIP ← Pop();
                tempEIP ← tempEIP AND 0000FFFFH;
                IF tempEIP not within code segment limits
                    THEN #GP(0); FI;
                EIP ← tempEIP;
```

```
                CS ← Pop(); (* 16-bit pop *)
            FI;
        IF instruction has immediate operand
            THEN
                SP ← SP + (SRC AND FFFFH); (* Release parameters from stack *)
        FI;
FI;
```

…

SCAS/SCASB/SCASW/SCASD—Scan String

…

SCAS, SCASB, SCASW, SCASD, and SCASQ can be preceded by the REP prefix for block comparisons of ECX bytes, words, doublewords, or quadwords. Often, however, these instructions will be used in a LOOP construct that takes some action based on the setting of status flags. See “REP/REPE/REPZ /REPNE/REPNZ—Repeat String Operation Prefix” in this chapter for a description of the REP prefix.

…

STOS/STOSB/STOSW/STOSD/STOSQ—Store String

…

The STOS, STOSB, STOSW, STOSD, STOSQ instructions can be preceded by the REP prefix for block loads of ECX bytes, words, or doublewords. More often, however, these instructions are used within a LOOP construct because data needs to be moved into the AL, AX, or EAX register before it can be stored. See “REP/REPE/REPZ /REPNE/REPNZ—Repeat String Operation Prefix” in this chapter for a description of the REP prefix.

…

XRSTOR—Restore Processor Extended States

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F AE /5	XRSTOR *mem*	A	Valid	Valid	Restore processor extended states from *memory*. The states are specified by EDX:EAX
REX.W+ 0F AE /5	XRSTOR64 *mem*	A	Valid	N.E.	Restore processor extended states from *memory*. The states are specified by EDX:EAX

…

XSAVE—Save Processor Extended States

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F AE /4	XSAVE *mem*	A	Valid	Valid	Save processor extended states to *memory*. The states are specified by EDX:EAX
REX.W+ 0F AE /4	XSAVE64 *mem*	A	Valid	N.E.	Save processor extended states to *memory*. The states are specified by EDX:EAX

…

6. Updates to Chapter 5, Volume 2B

Change bars show changes to Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B:* Instruction Set Reference, N-Z.

…

INVEPT— Invalidate Translations Derived from EPT

Opcode	Instruction	Description
66 0F 38 80	INVEPT r64, m128	Invalidates EPT-derived entries in the TLBs and paging-structure caches (in 64-bit mode)
66 0F 38 80	INVEPT r32, m128	Invalidates EPT-derived entries in the TLBs and paging-structure caches (outside 64-bit mode)

Description

Invalidates mappings in the translation lookaside buffers (TLBs) and paging-structure caches that were derived from **extended page tables** (EPT). (See Chapter 25, "Support for Address Translation" in *IA-32 Intel Architecture Software Developer's Manual, Volume 3B*.) Invalidation is based on the **INVEPT type** specified in the register operand and the **INVEPT descriptor** specified in the memory operand.

Outside IA-32e mode, the register operand is always 32 bits, regardless of the value of CS.D. In 64-bit mode, the register operand has 64 bits; however, if bits 63:32 of the register operand are not zero, INVEPT fails due to an attempt to use an unsupported INVEPT type (see below).

The INVEPT types supported by a logical processors are reported in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix "VMX Capability Reporting Facility" in *IA-32 Intel Architecture Software Developer's Manual, Volume 3B*). There are two INVEPT types currently defined:

- Single-context invalidation. If the INVEPT type is 1, the logical processor invalidates all mappings associated with bits 51:12 of the EPT pointer (EPTP) specified in the INVEPT descriptor. It may invalidate other mappings as well.
- Global invalidation: If the INVEPT type is 2, the logical processor invalidates mappings associated with all EPTPs.

If an unsupported INVEPT type is specified, the instruction fails.

INVEPT invalidates all the specified mappings for the indicated EPTP(s) regardless of the VPID and PCID values with which those mappings may be associated.

…

Operation

```
IF (not in VMX operation) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VM exit;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE
        INVEPT_TYPE ← value of register operand;
        IF IA32_VMX_EPT_VPID_CAP MSR indicates that processor does not support INVEPT_TYPE
            THEN VMfail(Invalid operand to INVEPT/INVVPID);
            ELSE      // INVEPT_TYPE must be 1 or 2
                INVEPT_DESC ← value of memory operand;
                EPTP ← INVEPT_DESC[63:0];
                CASE INVEPT_TYPE OF
                    1:                  // single-context invalidation
                        IF VM entry with the "enable EPT" VM execution control set to 1
                        would fail due to the EPTP value
                            THEN VMfail(Invalid operand to INVEPT/INVVPID);
                            ELSE
                                Invalidate mappings associated with EPTP[51:12];
                                VMsucceed;
                        FI;
                        BREAK;
```

```
            2:              // global invalidation
                Invalidate mappings associated with all EPTPs;
                VMsucceed;
                BREAK;
        ESAC;
    FI;
FI;
```

Flags Affected

See the operation section and Section 5.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains an unusable segment.
	If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory operand effective address is outside the SS segment limit.
	If the SS register contains an unusable segment.
#UD	If not in VMX operation.
	If the logical processor does not support EPT (IA32_VMX_PROCBASED_CTLS2[33]=0).
	If the logical processor supports EPT (IA32_VMX_PROCBASED_CTLS2[33]=1) but does not support the INVEPT instruction (IA32_VMX_EPT_VPID_CAP[20]=0).

Real-Address Mode Exceptions

#UD	A logical processor cannot be in real-address mode while in VMX operation and the INVEPT instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The INVEPT instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The INVEPT instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If not in VMX operation.

If the logical processor does not support EPT (IA32_VMX_PROCBASED_CTLS2[33]=0).

If the logical processor supports EPT (IA32_VMX_PROCBASED_CTLS2[33]=1) but does not support the INVEPT instruction (IA32_VMX_EPT_VPID_CAP[20]=0).

...

INVVPID— Invalidate Translations Based on VPID

Opcode	Instruction	Description
66 0F 38 81	INVVPID r64, m128	Invalidates entries in the TLBs and paging-structure caches based on VPID (in 64-bit mode)
66 0F 38 81	INVVPID r32, m128	Invalidates entries in the TLBs and paging-structure caches based on VPID (outside 64-bit mode)

Description

Invalidates mappings in the translation lookaside buffers (TLBs) and paging-structure caches based on **virtual-processor identifier** (VPID). (See Chapter 25, "Support for Address Translation" in *IA-32 Intel Architecture Software Developer's Manual, Volume 3B*.) Invalidation is based on the **INVVPID type** specified in the register operand and the **INVVPID descriptor** specified in the memory operand.

Outside IA-32e mode, the register operand is always 32 bits, regardless of the value of CS.D. In 64-bit mode, the register operand has 64 bits; however, if bits 63:32 of the register operand are not zero, INVVPID fails due to an attempt to use an unsupported INVVPID type (see below).

The INVVPID types supported by a logical processors are reported in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix "VMX Capability Reporting Facility" in *IA-32 Intel Architecture Software Developer's Manual, Volume 3B*). There are four INVVPID types currently defined:

- Individual-address invalidation: If the INVVPID type is 0, the logical processor invalidates mappings for a single linear address and tagged with the VPID specified in the INVVPID descriptor. In some cases, it may invalidate mappings for other linear addresses (or with other VPIDs) as well.
- Single-context invalidation: If the INVVPID type is 1, the logical processor invalidates all mappings tagged with the VPID specified in the INVVPID descriptor. In some cases, it may invalidate mappings for other VPIDs as well.
- All-contexts invalidation: If the INVVPID type is 2, the logical processor invalidates all mappings tagged with all VPIDs except VPID 0000H. In some cases, it may invalidate translations with VPID 0000H as well.
- Single-context invalidation, retaining global translations: If the INVVPID type is 3, the logical processor invalidates all mappings tagged with the VPID specified in the INVVPID descriptor except global translations. In some cases, it may invalidate global translations (and mappings with other VPIDs) as well. See the "Caching Translation Information" section in Chapter 4 of the *IA-32 Intel Architecture Software Developer's Manual, Volumes 3A* for information about global translations.

If an unsupported INVVPID type is specified, the instruction fails.

INVVPID invalidates all the specified mappings for the indicated VPID(s) regardless of the EPTP and PCID values with which those mappings may be associated.

…

Operation

```
IF (not in VMX operation) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VM exit;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE
        INVVPID_TYPE ← value of register operand;
        IF IA32_VMX_EPT_VPID_CAP MSR indicates that processor does not support
        INVVPID_TYPE
            THEN VMfail(Invalid operand to INVEPT/INVVPID);
            ELSE            // INVVPID_TYPE must be in the range 0–3
                INVVPID_DESC ← value of memory operand;
                IF INVVPID_DESC[63:16] ≠ 0
                    THEN VMfail(Invalid operand to INVEPT/INVVPID);
                    ELSE
                        CASE INVVPID_TYPE OF
                            0:                  // individual-address invalidation
                                VPID ← INVVPID_DESC[15:0];
                                IF VPID = 0
                                    THEN VMfail(Invalid operand to INVEPT/INVVPID);
                                    ELSE
                                        GL_ADDR ← INVVPID_DESC[127:64];
                                        IF (GL_ADDR is not in a canonical form)
                                            THEN
                                                VMfail(Invalid operand to INVEPT/INVVPID);
                                            ELSE
                                                Invalidate mappings for GL_ADDR tagged with
VPID;
                                                VMsucceed;
                                        FI;
                                FI;
                                BREAK;
                            1:                  // single-context invalidation
                                VPID_CTX ← INVVPID_DESC[15:0];
                                IF VPID = 0
                                    THEN VMfail(Invalid operand to INVEPT/INVVPID);
                                    ELSE
                                        Invalidate all mappings tagged with VPID;
                                        VMsucceed;
                                FI;
                                BREAK;
                            2:                  // all-context invalidation
                                Invalidate all mappings tagged with all non-zero VPIDs;
                                VMsucceed;
                                BREAK;
```

```
                3:              // single-context invalidation retaining globals
                    VPID ← INVVPID_DESC[15:0];
                    IF VPID = 0
                        THEN VMfail(Invalid operand to INVEPT/INVVPID);
                        ELSE
                            Invalidate all mappings tagged with VPID except global
translations;
                            VMsucceed;
                    FI;
                    BREAK;
            ESAC;
        FI;
    FI;
FI;
```

Flags Affected

See the operation section and Section 5.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains an unusable segment.
	If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory operand effective address is outside the SS segment limit.
	If the SS register contains an unusable segment.
#UD	If not in VMX operation.
	If the logical processor does not support VPIDs (IA32_VMX_PROCBASED_CTLS2[37]=0).
	If the logical processor supports VPIDs (IA32_VMX_PROCBASED_CTLS2[37]=1) but does not support the INVVPID instruction (IA32_VMX_EPT_VPID_CAP[32]=0).

Real-Address Mode Exceptions

#UD	A logical processor cannot be in real-address mode while in VMX operation and the INVVPID instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The INVVPID instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The INVVPID instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory destination operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If not in VMX operation.
	If the logical processor does not support VPIDs (IA32_VMX_PROCBASED_CTLS2[37]=0).
	If the logical processor supports VPIDs (IA32_VMX_PROCBASED_CTLS2[37]=1) but does not support the INVVPID instruction (IA32_VMX_EPT_VPID_CAP[32]=0).

...

5.4 VM INSTRUCTION ERROR NUMBERS

For certain error conditions, the VM-instruction error field is loaded with an error number to indicate the source of the error. Table Table 5-1 lists VM-instruction error numbers.

Table 5-1 VM-Instruction Error Numbers

Error Number	Description
1	VMCALL executed in VMX root operation
2	VMCLEAR with invalid physical address
3	VMCLEAR with VMXON pointer
4	VMLAUNCH with non-clear VMCS
5	VMRESUME with non-launched VMCS
6	VMRESUME after VMXOFF (VMXOFF and VMXON between VMLAUNCH and VMRESUME)[1]
7	VM entry with invalid control field(s)[2,3]
8	VM entry with invalid host-state field(s)[2]
9	VMPTRLD with invalid physical address
10	VMPTRLD with VMXON pointer
11	VMPTRLD with incorrect VMCS revision identifier
12	VMREAD/VMWRITE from/to unsupported VMCS component
13	VMWRITE to read-only VMCS component
15	VMXON executed in VMX root operation
16	VM entry with invalid executive-VMCS pointer[2]
17	VM entry with non-launched executive VMCS[2]
18	VM entry with executive-VMCS pointer not VMXON pointer (when attempting to deactivate the dual-monitor treatment of SMIs and SMM)[2]
19	VMCALL with non-clear VMCS (when attempting to activate the dual-monitor treatment of SMIs and SMM)
20	VMCALL with invalid VM-exit control fields
22	VMCALL with incorrect MSEG revision identifier (when attempting to activate the dual-monitor treatment of SMIs and SMM)
23	VMXOFF under dual-monitor treatment of SMIs and SMM
24	VMCALL with invalid SMM-monitor features (when attempting to activate the dual-monitor treatment of SMIs and SMM)
25	VM entry with invalid VM-execution control fields in executive VMCS (when attempting to return from SMM)[2,3]
26	VM entry with events blocked by MOV SS.
28	Invalid operand to INVEPT/INVVPID.

NOTES:

1. Earlier versions of this manual described this error as "VMRESUME with a corrupted VMCS".
2. VM-entry checks on control fields and host-state fields may be performed in any order. Thus, an indication by error number of one cause does not imply that there are not also other errors. Different processors may give different error numbers for the same VMCS.

3. Error number 7 is not used for VM entries that return from SMM that fail due to invalid VM-execution control fields in the executive VMCS. Error number 25 is used for these cases.

...

7. Updates to Appendix A, Volume 2B

Change bars show changes to Appendix A of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B:* Instruction Set Reference, N-Z.

...

A.2 KEY TO ABBREVIATIONS

Operands are identified by a two-character code of the form Zz. The first character, an uppercase letter, specifies the addressing method; the second character, a lowercase letter, specifies the type of operand.

A.2.1 Codes for Addressing Method

The following abbreviations are used to document addressing methods:

A Direct address: the instruction has no ModR/M byte; the address of the operand is encoded in the instruction. No base register, index register, or scaling factor can be applied (for example, far JMP (EA)).

C The reg field of the ModR/M byte selects a control register (for example, MOV (0F20, 0F22)).

D The reg field of the ModR/M byte selects a debug register (for example, MOV (0F21,0F23)).

E A ModR/M byte follows the opcode and specifies the operand. The operand is either a general-purpose register or a memory address. If it is a memory address, the address is computed from a segment register and any of the following values: a base register, an index register, a scaling factor, a displacement.

F EFLAGS/RFLAGS Register.

G The reg field of the ModR/M byte selects a general register (for example, AX (000)).

I Immediate data: the operand value is encoded in subsequent bytes of the instruction.

J The instruction contains a relative offset to be added to the instruction pointer register (for example, JMP (0E9), LOOP).

M The ModR/M byte may refer only to memory (for example, BOUND, LES, LDS, LSS, LFS, LGS, CMPXCHG8B).

N The R/M field of the ModR/M byte selects a packed-quadword, MMX technology register.

O The instruction has no ModR/M byte. The offset of the operand is coded as a word or double word (depending on address size attribute) in the instruction. No

base register, index register, or scaling factor can be applied (for example, MOV (A0–A3)).

P The reg field of the ModR/M byte selects a packed quadword MMX technology register.

Q A ModR/M byte follows the opcode and specifies the operand. The operand is either an MMX technology register or a memory address. If it is a memory address, the address is computed from a segment register and any of the following values: a base register, an index register, a scaling factor, and a displacement.

R The R/M field of the ModR/M byte may refer only to a general register (for example, MOV (0F20-0F23)).

S The reg field of the ModR/M byte selects a segment register (for example, MOV (8C,8E)).

U The R/M field of the ModR/M byte selects a 128-bit XMM register.

V The reg field of the ModR/M byte selects a 128-bit XMM register.

W A ModR/M byte follows the opcode and specifies the operand. The operand is either a 128-bit XMM register or a memory address. If it is a memory address, the address is computed from a segment register and any of the following values: a base register, an index register, a scaling factor, and a displacement.

X Memory addressed by the DS:rSI register pair (for example, MOVS, CMPS, OUTS, or LODS).

Y Memory addressed by the ES:rDI register pair (for example, MOVS, CMPS, INS, STOS, or SCAS).

A.2.2 Codes for Operand Type

The following abbreviations are used to document operand types:

a Two one-word operands in memory or two double-word operands in memory, depending on operand-size attribute (used only by the BOUND instruction).

b Byte, regardless of operand-size attribute.

c Byte or word, depending on operand-size attribute.

d Doubleword, regardless of operand-size attribute.

dq Double-quadword, regardless of operand-size attribute.

p 32-bit, 48-bit, or 80-bit pointer, depending on operand-size attribute.

pd 128-bit packed double-precision floating-point data.

pi Quadword MMX technology register (for example: mm0).

ps 128-bit packed single-precision floating-point data.

q Quadword, regardless of operand-size attribute.

s 6-byte or 10-byte pseudo-descriptor.

sd Scalar element of a 128-bit double-precision floating data.

ss Scalar element of a 128-bit single-precision floating data.

si Doubleword integer register (for example: eax).

v Word, doubleword or quadword (in 64-bit mode), depending on operand-size attribute.

w Word, regardless of operand-size attribute.

y Doubleword or quadword (in 64-bit mode), depending on operand-size attribute.

z Word for 16-bit operand-size or doubleword for 32 or 64-bit operand-size.

…

A.2.4.1 One-Byte Opcode Instructions

The opcode map for 1-byte opcodes is shown in Table Table A-2. The opcode map for 1-byte opcodes is arranged by row (the least-significant 4 bits of the hexadecimal value) and column (the most-significant 4 bits of the hexadecimal value). Each entry in the table lists one of the following types of opcodes:

- Instruction mnemonics and operand types using the notations listed in Section A.2
- Opcodes used as an instruction prefix

For each entry in the opcode map that corresponds to an instruction, the rules for interpreting the byte following the primary opcode fall into one of the following cases:

- A ModR/M byte is required and is interpreted according to the abbreviations listed in Section A.1 and Chapter 2, “Instruction Format,” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*. Operand types are listed according to notations listed in Section A.2.
- A ModR/M byte is required and includes an opcode extension in the reg field in the ModR/M byte. Use Table Table A-6 when interpreting the ModR/M byte.
- Use of the ModR/M byte is reserved or undefined. This applies to entries that represent an instruction prefix or entries for instructions without operands that use ModR/M (for example: 60H, PUSHA; 06H, PUSH ES).

…

Table A-2 One-byte Opcode Map: (00H — F7H) *

	0	1	2	3	4	5	6	7
0	ADD						PUSH ES[i64]	POP ES[i64]
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
1	ADC						PUSH SS[i64]	POP SS[i64]
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
2	AND						SEG=ES (Prefix)	DAA[i64]
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
3	XOR						SEG=SS (Prefix)	AAA[i64]
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
4	INC[i64] general register / REX[o64] Prefixes							
	eAX REX	eCX REX.B	eDX REX.X	eBX REX.XB	eSP REX.R	eBP REX.RB	eSI REX.RX	eDI REX.RXB
5	PUSH[d64] general register							
	rAX/r8	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
6	PUSHA[i64]/ PUSHAD[i64]	POPA[i64]/ POPAD[i64]	BOUND[i64] Gv, Ma	ARPL[i64] Ew, Gw MOVSXD[o64] Gv, Ev	SEG=FS (Prefix)	SEG=GS (Prefix)	Operand Size (Prefix)	Address Size (Prefix)
7	Jcc[f64], Jb - Short-displacement jump on condition							
	O	NO	B/NAE/C	NB/AE/NC	Z/E	NZ/NE	BE/NA	NBE/A
8	Immediate Grp 1[1A]				TEST		XCHG	
	Eb, Ib	Ev, Iz	Eb, Ib[i64]	Ev, Ib	Eb, Gb	Ev, Gv	Eb, Gb	Ev, Gv
9	NOP PAUSE(F3) XCHG r8, rAX	XCHG word, double-word or quad-word register with rAX						
		rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
A	MOV				MOVS/B Xb, Yb	MOVS/W/D/Q Xv, Yv	CMPS/B Xb, Yb	CMPS/W/D Xv, Yv
	AL, Ob	rAX, Ov	Ob, AL	Ov, rAX				
B	MOV immediate byte into byte register							
	AL/R8L, Ib	CL/R9L, Ib	DL/R10L, Ib	BL/R11L, Ib	AH/R12L, Ib	CH/R13L, Ib	DH/R14L, Ib	BH/R15L, Ib
C	Shift Grp 2[1A]		RETN[f64] Iw	RETN[f64]	LES[i64] Gz, Mp	LDS[i64] Gz, Mp	Grp 11[1A] - MOV	
	Eb, Ib	Ev, Ib					Eb, Ib	Ev, Iz
D	Shift Grp 2[1A]				AAM[i64] Ib	AAD[i64] Ib		XLAT/ XLATB
	Eb, 1	Ev, 1	Eb, CL	Ev, CL				
E	LOOPNE[f64]/ LOOPNZ[f64] Jb	LOOPE[f64]/ LOOPZ[f64] Jb	LOOP[f64] Jb	JrCXZ[f64]/ Jb	IN		OUT	
					AL, Ib	eAX, Ib	Ib, AL	Ib, eAX
F	LOCK (Prefix)		REPNE (Prefix)	REP/REPE (Prefix)	HLT	CMC	Unary Grp 3[1A]	
							Eb	Ev

Table A-2. One-byte Opcode Map: (08H — FFH) *

	8	9	A	B	C	D	E	F
0	OR						PUSH CS^{i64}	2-byte escape (Table A-3)
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
1	SBB						PUSH DS^{i64}	POP DS^{i64}
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
2	SUB						SEG=CS (Prefix)	DAS^{i64}
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
3	CMP						SEG=DS (Prefix)	AAS^{i64}
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz		
4	DEC^{i64} general register / REX^{o64} Prefixes							
	eAX REX.W	eCX REX.WB	eDX REX.WX	eBX REX.WXB	eSP REX.WR	eBP REX.WRB	eSI REX.WRX	eDI REX.WRXB
5	POP^{d64} into general register							
	rAX/r8	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
6	PUSH^{d64} Iz	IMUL Gv, Ev, Iz	PUSH^{d64} Ib	IMUL Gv, Ev, Ib	INS/ INSB Yb, DX	INS/ INSW/ INSD Yz, DX	OUTS/ OUTSB DX, Xb	OUTS/ OUTSW/ OUTSD DX, Xz
7	Jcc^{f64}, Jb- Short displacement jump on condition							
	S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
8	MOV				MOV Ev, Sw	LEA Gv, M	MOV Sw, Ew	Grp 1A^{1A} POP^{d64} Ev
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev				
9	CBW/ CWDE/ CDQE	CWD/ CDQ/ CQO	CALLF^{i64} Ap	FWAIT/ WAIT	PUSHF/D/Q^{d64}/ Fv	POPF/D/Q^{d64}/ Fv	SAHF	LAHF
A	TEST		STOS/B Yb, AL	STOS/W/D/Q Yv, rAX	LODS/B AL, Xb	LODS/W/D/Q rAX, Xv	SCAS/B AL, Yb	SCAS/W/D/Q rAX, Xv
	AL, Ib	rAX, Iz						
B	MOV immediate word or double into word, double, or quad register							
	rAX/r8, Iv	rCX/r9, Iv	rDX/r10, Iv	rBX/r11, Iv	rSP/r12, Iv	rBP/r13, Iv	rSI/r14, Iv	rDI/r15 , Iv
C	ENTER Iw, Ib	LEAVE^{d64}	RETF Iw	RETF	INT 3	INT Ib	INTO^{i64}	IRET/D/Q
D	ESC (Escape to coprocessor instruction set)							
E	CALL^{f64} Jz	JMP			IN		OUT	
		near^{f64} Jz	far^{i64} AP	short^{f64} Jb	AL, DX	eAX, DX	DX, AL	DX, eAX
F	CLC	STC	CLI	STI	CLD	STD	INC/DEC Grp 4^{1A}	INC/DEC Grp 5^{1A}

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-3 Two-byte Opcode Map: 00H — 77H (First Byte is 0FH) *

	pfx	0	1	2	3	4	5	6	7
0		Grp 6[1A]	Grp 7[1A]	LAR Gv, Ew	LSL Gv, Ew		SYSCALL[064]	CLTS	SYSRET[064]
1		movups	movups	movlps Vq, Mq movhlps Vq, Uq	movlps Mq, Vq	unpcklps Vps, Wq	unpckhps Vps, Wq	movhps Vq, Mq movlhps Vq, Uq	movhps Mq, Vq
	66	movupd	movupd Wpd,Vpd	movlpd Vq, Mq	movlpd Mq, Vq	unpcklpd	unpckhpd	movhpd Vq, Mq	movhpd Mq, Vq
	F3	movss Vss, Wss	movss Wss, Vss	movsldup				movshdup	
	F2	movsd Vsd, Wsd	movsd Vsd, Wsd	movddup					
2	2	MOV Rd, Cd	MOV Rd, Dd	MOV Cd, Rd	MOV Dd, Rd				
3	3	WRMSR	RDTSC	RDMSR	RDPMC	SYSENTER	SYSEXIT		GETSEC
4	4	CMOVcc, (Gv, Ev) - Conditional Move							
		O	NO	B/C/NAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
5		movmskps Gy, U	sqrtps	rsqrtps	rcpps	andps Vps, Wps	andnps Vps, Wps	orps Vps, Wps	xorps Vps, Wps
	66	movmskpd Gy,U	sqrtpd Wpd,Vpd			andpd Wpd, Vpd	andnpd Wpd, Vpd	orpd Wpd, Vpd	xorpd Wpd, Vpd
	F3		sqrtss Vss, Wss	rsqrtss Vss, Wss	rcpss Vss, Wss				
	F2		sqrtsd Vsd, Wsd						
6		punpcklbw Pq, Qd	punpcklwd Pq, Qd	punpckldq Pq, Qd	packsswb Pq, Qq	pcmpgtb Pq, Qq	pcmpgtw Pq, Qq	pcmpgtd Pq, Qq	packuswb Pq, Qq
	66	punpcklbw Vdq, Wdq	punpcklwd Vdq, Wdq	punpckldq Vdq, Wdq	packsswb Vdq, Wdq	pcmpgtb Vdq, Wdq	pcmpgtw Vdq, Wdq	pcmpgtd Vdq, Wdq	packuswb Vdq, Wdq
	F3								
7		pshufw Pq, Qq, Ib	(Grp 12[1A])	(Grp 13[1A])	(Grp 14[1A])	pcmpeqb Pq, Qq	pcmpeqw Pq, Qq	pcmpeqd Pq, Qq	emms
	66	pshufd Vdq,Wdq,Ib				pcmpeqb Vdq, Wdq	pcmpeqw Vdq, Wdq	pcmpeqd Vdq, Wdq	
	F3	pshufhw Vdq,Wdq,Ib							
	F2	pshuflw Vdq,Wdq,Ib							

Table A-3. Two-byte Opcode Map: 08H — 7FH (First Byte is 0FH) *

	pfx	8	9	A	B	C	D	E	F
0		INVD	WBINVD		2-byte Illegal Opcodes UD2[1B]		NOP Ev		
1		Prefetch[1C] (Grp 16[1A])							NOP Ev
2		movaps Vps, Wps	movaps Wps, Vps	cvtpi2ps Vps, Qpi	movntps Mps, Vps	cvttps2pi Ppi, Wps	cvtps2pi Ppi, Wps	ucomiss Vss, Wss	comiss Vss, Wss
	66	movapd Vpd, Wpd	movapd Wpd,Vpd	cvtpi2pd Vpd, Qpi	movntpd Mpd, Vpd	cvttpd2pi Ppi, Wpd	cvtpd2pi Qpi, Wpd	ucomisd Vsd, Wsd	comisd Vsd, Wsd
	F3			cvtsi2ss Vss, Ey		cvttss2si Gy, Wss	cvtss2si Gy, Wss		
	F2			cvtsi2sd Vsd, Ey		cvttsd2si Gy, Wsd	cvtsd2si Gy, Wsd		
3	3	3-byte escape (Table A-4)		3-byte escape (Table A-5)					
4	4	CMOVcc(Gv, Ev) - Conditional Move							
		S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
5		addps Vps, Wps	mulps Vps, Wps	cvtps2pd	cvtdq2ps	subps Vps, Wps	minps Vps, Wps	divps Vps, Wps	maxps Vps, Wps
	66	addpd Vpd, Wpd	mulpd Vpd, Wpd	cvtpd2ps Vps, Wpd	cvtps2dq Vdq, Wps	subpd Vpd, Wpd	minpd Vpd, Wpd	divpd Vpd, Wpd	maxpd Vpd, Wpd
	F3	addss Vss, Wss	mulss Vss, Wss	cvtss2sd Vsd, Wss	cvttps2dq Vdq, Wps	subss Vss, Wss	minss Vss, Wss	divss Vss, Wss	maxss Vss, Wss
	F2	addsd Vsd, Wsd	mulsd Vsd, Wsd	cvtsd2ss Vss, Wsd		subsd Vsd, Wsd	minsd Vsd, Wsd	divsd Vsd, Wsd	maxsd Vsd, Wsd
6		punpckhbw Pq, Qd	punpckhwd Pq, Qd	punpckhdq Pq, Qd	packssdw Pq, Qd			movd/q Pd, Ey	movq Pq, Qq
	66	punpckhbw Vdq, Wdq	punpckhwd Vdq, Wdq	punpckhdq Vdq, Wdq	packssdw Vdq, Wdq	punpcklqdq Vdq, Wdq	punpckhqdq Vdq, Wdq	movd/q Vy, Ey	movdqa
	F3								movdqu
7		VMREAD Ey, Gy	VMWRITE Gy, Ey					movd/q Ey, Pd	movq Qq, Pq
	66					haddpd Vpd, Wpd	hsubpd Vpd, Wpd	movd/q Ey, Vy	movdqa
	F3							movq Vq, Wq	movdqu
	F2					haddps Vps, Wps	hsubps Vps, Wps		

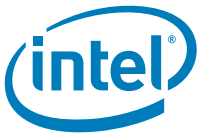

Table A-3. Two-byte Opcode Map: 80H — F7H (First Byte is 0FH) *

	pfx	0	1	2	3	4	5	6	7
8		Jcc[f64], Jz - Long-displacement jump on condition							
		O	NO	B/CNAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
9		SETcc, Eb - Byte Set on condition							
		O	NO	B/C/NAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
A		PUSH[d64] FS	POP[d64] FS	CPUID	BT Ev, Gv	SHLD Ev, Gv, Ib	SHLD Ev, Gv, CL		
B		CMPXCHG		LSS Gv, Mp	BTR Ev, Gv	LFS Gv, Mp	LGS Gv, Mp	MOVZX	
		Eb, Gb	Ev, Gv					Gv, Eb	Gv, Ew
C		XADD Eb, Gb	XADD Ev, Gv	cmpps Vps,Wps,Ib	movnti My, Gy	pinsrw Pq,Ry/Mw,Ib	pextrw Gd, Nq, Ib	shufps Vps,Wps,Ib	Grp 9[1A]
	66			cmppd Vpd,Wpd,Ib		pinsrw Vdq,Ry/Mw,Ib	pextrw Gd, Udq, Ib	shufpd Vpd,Wpd,Ib	
	F3			cmpss Vss,Wss,Ib					
	F2			cmpsd Vsd,Wsd,Ib					
D			psrlw Pq, Qq	psrld Pq, Qq	psrlq Pq, Qq	paddq Pq, Qq	pmullw Pq, Qq		pmovmskb Gd, Nq
	66	addsubpd Vpd, Wpd	psrlw Vdq, Wdq	psrld Vdq, Wdq	psrlq Vdq, Wdq	paddq Vdq, Wdq	pmullw Vdq, Wdq	movq Wq, Vq	pmovmskb Gd, Udq
	F3							movq2dq Vdq, Nq	
	F2	addsubps Vps, Wps						movdq2q Pq, Uq	
E		pavgb Pq, Qq	psraw Pq, Qq	psrad Pq, Qq	pavgw Pq, Qq	pmulhuw Pq, Qq	pmulhw Pq, Qq		movntq Mq, Pq
	66	pavgb Vdq, Wdq	psraw Vdq, Wdq	psrad Vdq, Wdq	pavgw Vdq, Wdq	pmulhuw Vdq, Wdq	pmulhw Vdq, Wdq	cvttpd2dq	movntdq
	F3							cvtdq2pd	
	F2							cvtpd2dq	
F			psllw Pq, Qq	pslld Pq, Qq	psllq Pq, Qq	pmuludq Pq, Qq	pmaddwd Pq, Qq	psadbw Pq, Qq	maskmovq Pq, Nq
	66		psllw Vdq, Wdq	pslld Vdq, Wdq	psllq Vdq, Wdq	pmuludq Vdq, Wdq	pmaddwd Vdq, Wdq	psadbw Vdq, Wdq	maskmovdqu Vdq, Udq
	F2	lddqu							

Table A-3. Two-byte Opcode Map: 88H — FFH (First Byte is 0FH) *

	pfx	8	9	A	B	C	D	E	F
8		Jcc[f64], Jz - Long-displacement jump on condition							
		S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
9		SETcc, Eb - Byte Set on condition							
		S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
A		PUSH[d64] GS	POP[d64] GS	RSM	BTS Ev, Gv	SHRD Ev, Gv, Ib	SHRD Ev, Gv, CL	(Grp 15[1A])[1C]	IMUL Gv, Ev
B		*JMPE* (reserved for emulator on IPF)	Grp 10[1A] Invalid Opcode[1B]	Grp 8[1A] Ev, Ib	BTC Ev, Gv	BSF Gv, Ev	BSR Gv, Ev	MOVSX	
								Gv, Eb	Gv, Ew
	F3	POPCNT Gv, Ev							
C		BSWAP							
		RAX/EAX/ R8/R8D	RCX/ECX/ R9/R9D	RDX/EDX/ R10/R10D	RBX/EBX/ R11/R11D	RSP/ESP/ R12/R12D	RBP/EBP/ R13/R13D	RSI/ESI/ R14/R14D	RDI/EDI/ R15/R15D
D		psubusb Pq, Qq	psubusw Pq, Qq	pminub Pq, Qq	pand Pq, Qq	paddusb Pq, Qq	paddusw Pq, Qq	pmaxub Pq, Qq	pandn Pq, Qq
	66	psubusb Vdq, Wdq	psubusw Vdq, Wdq	pminub Vdq, Wdq	pand Vdq, Wdq	paddusb Vdq, Wdq	paddusw Vdq, Wdq	pmaxub Vdq, Wdq	pandn Vdq, Wdq
	F3								
	F2								
E		psubsb Pq, Qq	psubsw Pq, Qq	pminsw Pq, Qq	por Pq, Qq	paddsb Pq, Qq	paddsw Pq, Qq	pmaxsw Pq, Qq	pxor Pq, Qq
	66	psubsb Vdq, Wdq	psubsw Vdq, Wdq	pminsw Vdq, Wdq	por Vdq, Wdq	paddsb Vdq, Wdq	paddsw Vdq, Wdq	pmaxsw Vdq, Wdq	pxor Vdq, Wdq
	F3								
	F2								
F		psubb Pq, Qq	psubw Pq, Qq	psubd Pq, Qq	psubq Pq, Qq	paddb Pq, Qq	paddw Pq, Qq	paddd Pq, Qq	
	66	psubb Vdq, Wdq	psubw Vdq, Wdq	psubd Vdq, Wdq	psubq Vdq, Wdq	paddb Vdq, Wdq	paddw Vdq, Wdq	paddd Vdq, Wdq	
	F2								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

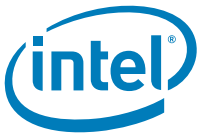

Table A-4 Three-byte Opcode Map: 00H — F7H (First Two Bytes are 0F 38H) *

	pfx	0	1	2	3	4	5	6	7
0		pshufb Pq, Qq	phaddw Pq, Qq	phaddd Pq, Qq	phaddsw Pq, Qq	pmaddubsw Pq, Qq	phsubw Pq, Qq	phsubd Pq, Qq	phsubsw Pq, Qq
	66	pshufb Vdq, Wdq	phaddw Vdq, Wdq	phaddd Vdq, Wdq	phaddsw Vdq, Wdq	pmaddubsw Vdq, Wdq	phsubw Vdq, Wdq	phsubd Vdq, Wdq	phsubsw Vdq, Wdq
1	66	pblendvb Vdq, Wdq				blendvps	blendvpd		ptest
2	66	pmovsxbw Vdq, Udq/Mq	pmovsxbd Vdq, Udq/Md	pmovsxbq Vdq, Udq/Mw	pmovsxwd Vdq, Udq/Mq	pmovsxwq Vdq, Udq/Md	pmovsxdq Vdq, Udq/Mq		
3	66	pmovzxbw Vdq, Udq/Mq	pmovzxbd Vdq, Udq/Md	pmovzxbq Vdq, Udq/Mw	pmovzxwd Vdq, Udq/Mq	pmovzxwq Vdq, Udq/Md	pmovzxdq Vdq, Udq/Mq		pcmpgtq Vdq, Wdq
4	66	pmulld Vdq, Wdq	phminposuw Vdq, Wdq						
5									
6									
7									
8	66	INVEPT Gy, Mdq	INVVPID Gy, Mdq						
9									
A									
B									
C									
D									
E									
F		MOVBE Gy, My	MOVBE My, Gy						
	66	MOVBE Gw, Mw	MOVBE Mw, Gw						
	F3								
	F2	CRC32 Gd, Eb	CRC32 Gd, Ey						
	66 & F2	CRC32 Gd, Eb	CRC32 Gd, Ew						

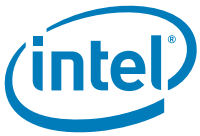

Table A-4. Three-byte Opcode Map: 08H — FFH (First Two Bytes are 0F 38H) *

	pfx	8	9	A	B	C	D	E	F
0		psignb Pq, Qq	psignw Pq, Qq	psignd Pq, Qq	pmulhrsw Pq, Qq				
	66	psignb Vdq, Wdq	psignw Vdq, Wdq	psignd Vdq, Wdq	pmulhrsw Vdq, Wdq				
1						pabsb Pq, Qq	pabsw Pq, Qq	pabsd Pq, Qq	
	66					pabsb Vdq, Wdq	pabsw Vdq, Wdq	pabsd Vdq, Wdq	
2	66	pmuldq Vdq, Wdq	pcmpeqq Vdq, Wdq	movntdqa Vdq, Mdq	packusdw Vdq, Wdq				
3	66	pminsb Vdq, Wdq	pminsd Vdq, Wdq	pminuw Vdq, Wdq	pminud Vdq, Wdq	pmaxsb Vdq, Wdq	pmaxsd Vdq, Wdq	pmaxuw Vdq, Wdq	pmaxud Vdq, Wdq
4									
5									
6									
7									
8									
9									
A									
B									
C									
D	66				AESIMC Vdq, Wdq	AESENC Vdq,Wdq	AESENCLAST Vdq,Wdq	AESDEC Vdq,Wdq	AESDECLAST Vdq,Wdq
E									
F									
	66								
	F3								
	F2								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-5 Three-byte Opcode Map: 00H — F7H (First two bytes are 0F 3AH) *

	pfx	0	1	2	3	4	5	6	7
0									
1	66					pextrb Rd/Mb, Vdq, Ib	pextrw Rd/Mw, Vdq, Ib	pextrd/q Ey, Vdq, Ib	extractps Ed, Vdq, Ib
2	66	pinsrb Vdq,Ry/Mb,Ib	insertps Vdq,Udq/Md,Ib	pinsrd/q Vdq,Ey,Ib					
3									
4	66	dpps	dppd	mpsadbw Vdq,Wdq,Ib		pclmulqdq Vdq,Wdq,Ib			
5									
6	66	pcmpestrm dq, Wdq, Ib	pcmpestri Vdq, Wdq, Ib	pcmpistrm Vdq, Wdq, Ib	pcmpistri Vdq, Wdq, Ib				
7									
8									
9									
A									
B									
C									
D									
E									
F									

Table A-5. Three-byte Opcode Map: 08H — FFH (First Two Bytes are 0F 3AH) *

	pfx	8	9	A	B	C	D	E	F
0									palignr Pq, Qq, Ib
	66	roundps	roundpd	roundss Vss,Wss,Ib	roundsd Vss,Wss,Ib	blendps	blendpd	pblendw Vdq,Wdq,Ib	palignr Vdq,Wdq,Ib
1									
2									
3									
4									
5									
6									
7									
8									
9									
A									
B									
C									
D	66								AESKEYGEN Vdq, Wdq, Ib
E									
F									

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

…

A.4.2 Opcode Extension Tables

See Table Table A-6 below.

Table A-6 Opcode Extensions for One- and Two-byte Opcodes by Group Number *

Opcode	Group	Mod 7,6	pfx	Encoding of Bits 5,4,3 of the ModR/M Byte (bits 2,1,0 in parenthesis)							
				000	001	010	011	100	101	110	111
80-83	1	mem, 11B		ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
8F	1A	mem, 11B		POP							
C0,C1 reg, imm D0, D1 reg, 1 D2, D3 reg, CL	2	mem, 11B		ROL	ROR	RCL	RCR	SHL/SAL	SHR		SAR
F6, F7	3	mem, 11B		TEST Ib/Iz		NOT	NEG	MUL AL/rAX	IMUL AL/rAX	DIV AL/rAX	IDIV AL/rAX
FE	4	mem, 11B		INC Eb	DEC Eb						
FF	5	mem, 11B		INC Ev	DEC Ev	CALLN[f64] Ev	CALLF Ep	JMPN[f64] Ev	JMPF Ep	PUSH[d64] Ev	
0F 00	6	mem, 11B		SLDT Rv/Mw	STR Rv/Mw	LLDT Ew	LTR Ew	VERR Ew	VERW Ew		
0F 01	7	mem		SGDT Ms	SIDT Ms	LGDT Ms	LIDT Ms	SMSW Mw/Rv		LMSW Ew	INVLPG Mb
		11B		VMCALL (001) VMLAUNCH (010) VMRESUME (011) VMXOFF (100)	MONITOR (000) MWAIT (001)	XGETBV (000) XSETBV (001)					SWAPGS [o64](000) RDTSCP (001)
0F BA	8	mem, 11B						BT	BTS	BTR	BTC
0F C7	9	mem			CMPXCH8B Mq CMPXCHG16B Mdq					VMPTRLD Mq	VMPTRST Mq
			66							VMCLEAR Mq	
			F3							VMXON Mq	VMPTRST Mq
		11B									
0F B9	10	mem									
		11B									
C6	11	mem, 11B		MOV Eb, Ib							
C7		mem		MOV Ev, Iz							
		11B									

Table A-6 Opcode Extensions for One- and Two-byte Opcodes by Group Number * (Continued)

Opcode	Group	Mod 7,6	pfx	Encoding of Bits 5,4,3 of the ModR/M Byte (bits 2,1,0 in parenthesis)							
				000	001	010	011	100	101	110	111
0F 71	12	mem									
		11B				psrlw Nq, Ib		psraw Nq, Ib		psllw Nq, Ib	
			66			psrlw Udq,Ib		psraw Udq,Ib		psllw Udq,Ib	
0F 72	13	mem									
		11B				psrld Nq, Ib		psrad Nq, Ib		pslld Nq, Ib	
			66			psrld Udq,Ib		psrad Udq,Ib		pslld Udq,Ib	
0F 73	14	mem									
		11B				psrlq Nq, Ib				psllq Nq, Ib	
			66			psrlq Udq,Ib	psrldq Udq,Ib			psllq Udq,Ib	pslldq Udq,Ib
0F AE	15	mem		fxsave	fxrstor	ldmxcsr	stmxcsr	XSAVE	XRSTOR		clflush
		11B							lfence	mfence	sfence
0F 18	16	mem		prefetch NTA	prefetch T0	prefetch T1	prefetch T2				
		11B									

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

...

8. Updates to Appendix B, Volume 2B

Change bars show changes to Appendix B of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B:* Instruction Set Reference, N-Z.

...

B.12 AESNI AND PCLMULQDQ INSTRUCTION FORMATS AND ENCODINGS

Table Table B-33 shows the formats and encodings for AESNI and PCLMULQDQ instructions.

Table B-33 Formats and Encodings of AESNI and PCLMULQDQ Instructions

Instruction and Format	Encoding
AESDEC—Perform One Round of an AES Decryption Flow	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000:1101 1110:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000:1101 1110: mod xmmreg r/m
AESDECLAST—Perform Last Round of an AES Decryption Flow	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000:1101 1111:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000:1101 1111: mod xmmreg r/m
AESENC—Perform One Round of an AES Encryption Flow	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000:1101 1100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000:1101 1100: mod xmmreg r/m
AESENCLAST—Perform Last Round of an AES Encryption Flow	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000:1101 1101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000:1101 1101: mod xmmreg r/m
AESIMC—Perform the AES InvMixColumn Transformation	
xmmreg2 to xmmreg1	0110 0110:0000 1111:0011 1000:1101 1011:11 xmmreg1 xmmreg2
mem to xmmreg1	0110 0110:0000 1111:0011 1000:1101 1011: mod xmmreg r/m

Instruction and Format	Encoding
AESKEYGENASSIST—AES Round Key Generation Assist	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010:1101 1111:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010:1101 1111: mod xmmreg r/m: imm8
PCLMULQDQ—Carry-Less Multiplication Quadword	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010:0100 0100:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010:0100 0100: mod xmmreg r/m: imm8

…

9. Updates to Appendix C, Volume 2B

Change bars show changes to Appendix C of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B:* Instruction Set Reference, N-Z.

…

Table C-1 Simple Intrinsics

Mnemonic	Intrinsic
…	
AESDEC	__m128i _mm_aesdec (__m128i, __m128i)
AESDECLAST	__m128i _mm_aesdeclast (__m128i, __m128i)
AESENC	__m128i _mm_aesenc (__m128i, __m128i)
AESENCLAST	__m128i _mm_aesenclast (__m128i, __m128i)
AESIMC	__m128i _mm_aesimc (__m128i)
AESKEYGENASSIST	__m128i _mm_aesimc (__m128i, const int)
…	
PCLMULQDQ	__m128i _mm_clmulepi64_si128 (__m128i, __m128i, const int)
…	

…

10. Updates to Chapter 2, Volume 3A

Change bars show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

…

2.1.6.1 System Registers in IA-32e Mode

In IA-32e mode, the four system-descriptor-table registers (GDTR, IDTR, LDTR, and TR) are expanded in hardware to hold 64-bit base addresses. EFLAGS becomes the 64-bit RFLAGS register. CR0–CR4 are expanded to 64 bits. CR8 becomes available. CR8 provides read-write access to the task priority register (TPR) so that the operating system can control the priority classes of external interrupts.

In 64-bit mode, debug registers DR0–DR7 are 64 bits. In compatibility mode, address-matching in DR0–DR3 is also done at 64-bit granularity.

...

OSFXSR

Operating System Support for FXSAVE and FXRSTOR instructions (bit 9 of CR4) — When set, this flag: (1) indicates to software that the operating system supports the use of the FXSAVE and FXRSTOR instructions, (2) enables the FXSAVE and FXRSTOR instructions to save and restore the contents of the XMM and MXCSR registers along with the contents of the x87 FPU and MMX registers, and (3) enables the processor to execute SSE/SSE2/SSE3/SSSE3/SSE4 instructions, with the exception of the PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

If this flag is clear, the FXSAVE and FXRSTOR instructions will save and restore the contents of the x87 FPU and MMX instructions, but they may not save and restore the contents of the XMM and MXCSR registers. Also, the processor will generate an invalid opcode exception (#UD) if it attempts to execute any SSE/SSE2/SSE3 instruction, with the exception of PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. The operating system or executive must explicitly set this flag.

NOTE

CPUID feature flags FXSR indicates availability of the FXSAVE/FXRSTOR instructions. The OSFXSR bit provides operating system software with a means of enabling FXSAVE/FXRSTOR to save/restore the contents of the X87 FPU, XMM and MXCSR registers. Consequently OSFXSR bit indicates that the operating system provides context switch support for SSE/SSE2/SSE3/SSSE3/SSE4.

OSXMMEXCPT

Operating System Support for Unmasked SIMD Floating-Point Exceptions (bit 10 of CR4) — When set, indicates that the operating system supports the handling of unmasked SIMD floating-point exceptions through an exception handler that is invoked when a SIMD floating-point exception (#XF) is generated. SIMD floating-point exceptions are only generated by SSE/SSE2/SSE3/SSE4.1 SIMD floating-point instructions.

The operating system or executive must explicitly set this flag. If this flag is not set, the processor will generate an invalid opcode exception (#UD) whenever it detects an unmasked SIMD floating-point exception.

VMXE

VMX-Enable Bit (bit 13 of CR4) — Enables VMX operation when set. See Chapter 20, “Introduction to Virtual-Machine Extensions.”

SMXE

SMX-Enable Bit (bit 14 of CR4) — Enables SMX operation when set. See

Chapter 6, “Safer Mode Extensions Reference” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*.

PCIDE

PCID-Enable Bit (bit 17 of CR4) — Enables process-context identifiers (PCIDs) when set. See Section 4.10.1, “Process-Context Identifiers (PCIDs)”. Can be set only in IA-32e mode (if IA32_EFER.LMA = 1).

…

2.5.1 CPUID Qualification of Control Register Flags

Not all flags in control register CR4 are implemented on all processors. With the exception of the PCE flag, they can be qualified with the CPUID instruction to determine if they are implemented on the processor before they are used.

The CR8 register is available on processors that support Intel 64 architecture.

…

•

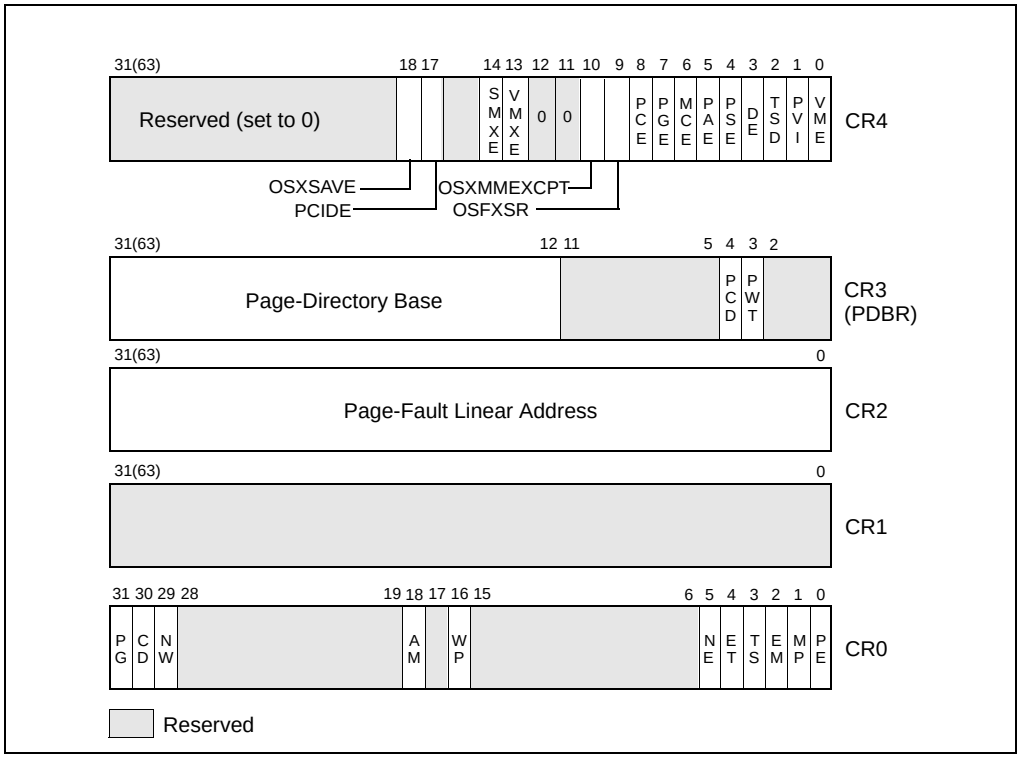

Figure 2-6 Control Registers

…

11. Updates to Chapter 4, Volume 3A

Change bars show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

4.1 PAGING MODES AND CONTROL BITS

Paging behavior is controlled by the following control bits:

- The WP and PG flags in control register CR0 (bit 16 and bit 31, respectively).
- The PSE, PAE, PGE, and PCIDE flags in control register CR4 (bit 4, bit 5, bit 7, and bit 17, respectively).
- The LME and NXE flags in the IA32_EFER MSR (bit 8 and bit 11, respectively).

Software enables paging by using the MOV to CR0 instruction to set CR0.PG. Before doing so, software should ensure that control register CR3 contains the physical address of the first paging structure that the processor will use for linear-address translation (see Section 4.2) and that structure is initialized as desired. See Table 4-3, Table 4-7, and Table Table 4-12 for the use of CR3 in the different paging modes.

Section 4.1.1 describes how the values of CR0.PG, CR4.PAE, and IA32_EFER.LME determine whether paging is in use and, if so, which of three paging modes is in use. Section 4.1.2 explains how to manage these bits to establish or make changes in paging modes. Section 4.1.3 discusses how CR0.WP, CR4.PSE, CR4.PGE, CR4.PCIDE, and IA32_EFER.NXE modify the operation of the different paging modes.

4.1.1 Three Paging Modes

If CR0.PG = 0, paging is not used. The logical processor treats all linear addresses as if they were physical addresses. CR4.PAE and IA32_EFER.LME are ignored by the processor, as are CR0.WP, CR4.PSE, and CR4.PGE, and IA32_EFER.NXE.

Paging is enabled if CR0.PG = 1. Paging can be enabled only if protection is enabled (CR0.PE = 1). If paging is enabled, one of three paging modes is used. The values of CR4.PAE and IA32_EFER.LME determine which paging mode is used:

- If CR0.PG = 1 and CR4.PAE = 0, **32-bit paging** is used. 32-bit paging is detailed in Section 4.3. 32-bit paging uses CR0.WP, CR4.PSE, and CR4.PGE as described in Section 4.1.3.
- If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0, **PAE paging** is used. PAE paging is detailed in Section 4.4. PAE paging uses CR0.WP, CR4.PGE, and IA32_EFER.NXE as described in Section 4.1.3.
- If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 1, **IA-32e paging** is used.[1] IA-32e paging is detailed in Section 4.5. IA-32e paging uses CR0.WP, CR4.PGE,

1. The LMA flag in the IA32_EFER MSR (bit 10) is a status bit that indicates whether the logical processor is in IA-32e mode (and thus using IA-32e paging). The processor always sets IA32_EFER.LMA to CR0.PG & IA32_EFER.LME. Software cannot directly modify IA32_EFER.LMA; an execution of WRMSR to the IA32_EFER MSR ignores bit 10 of its source operand.

CR4.PCIDE, and IA32_EFER.NXE as described in Section 4.1.3. IA-32e paging is available only on processors that support the Intel 64 architecture.

…

4.1.2 Paging-Mode Enabling

If CR0.PG = 1, a logical processor is in one of three paging modes, depending on the values of CR4.PAE and IA32_EFER.LME. Figure 4-1 illustrates how software can enable these modes and make transitions between them. The following items identify certain limitations and other details:

…

- CR4.PAE cannot be cleared while IA-32e paging is active (CR0.PG = 1 and IA32_EFER.LME = 1). Attempts to do so using MOV to CR4 cause a general-protection exception (#GP(0)).
- Regardless of the current paging mode, software can disable paging by clearing CR0.PG with MOV to CR0.[1]
- Software can make transitions between 32-bit paging and PAE paging by changing the value of CR4.PAE with MOV to CR4.

…

4.1.3 Paging-Mode Modifiers

Details of how each paging mode operates are determined by the following control bits:

- The WP flag in CR0 (bit 16).
- The PSE, PGE, and PCIDE flags in CR4 (bit 4, bit 7, and bit 17, respectively).
- The NXE flag in the IA32_EFER MSR (bit 11).

CR0.WP allows pages to be protected from supervisor-mode writes. If CR0.WP = 0, software operating with CPL < 3 (supervisor mode) can write to linear addresses with read-only access rights; if CR0.WP = 1, it cannot. (Software operating with CPL = 3 — user mode — cannot write to linear addresses with read-only access rights, regardless of the value of CR0.WP.) Section 4.6 explains how access rights are determined.

CR4.PSE enables 4-MByte pages for 32-bit paging. If CR4.PSE = 0, 32-bit paging can use only 4-KByte pages; if CR4.PSE = 1, 32-bit paging can use both 4-KByte pages and 4-MByte pages. See Section 4.3 for more information. (PAE paging and IA-32e paging can use multiple page sizes regardless of the value of CR4.PSE.)

CR4.PGE enables global pages. If CR4.PGE = 0, no translations are shared across address spaces; if CR4.PGE = 1, specified translations may be shared across address spaces. See Section 4.10.2.4 for more information.

CR4.PCIDE enables process-context identifiers (PCIDs) for IA-32e paging (CR4.PCIDE can be 1 only when IA-32e paging is in use). PCIDs allow a logical processor to cache information for multiple linear-address spaces. See Section 4.10.1 for more information.

…

1. If CR4.PCIDE = 1, an attempt to clear CR0.PG causes a general-protection exception (#GP); software should clear CR4.PCIDE before attempting to disable paging.

4.1.4 Enumeration of Paging Features by CPUID

Software can discover support for different paging features using the CPUID instruction:

- PSE: page-size extensions for 32-bit paging.
  If CPUID.01H:EDX.PSE [bit 3] = 1, CR4.PSE may be set to 1, enabling support for 4-MByte pages with 32-bit paging (see Section 4.3).
- PAE: physical-address extension.
  If CPUID.01H:EDX.PAE [bit 6] = 1, CR4.PAE may be set to 1, enabling PAE paging (this setting is also required for IA-32e paging).
- PGE: global-page support.
  If CPUID.01H:EDX.PGE [bit 13] = 1, CR4.PGE may be set to 1, enabling the global-page feature (see Section 4.10.2.4).
- PAT: page-attribute table.
  If CPUID.01H:EDX.PAT [bit 16] = 1, the 8-entry page-attribute table (PAT) is supported. When the PAT is supported, three bits in certain paging-structure entries select a memory type (used to determine type of caching used) from the PAT (see Section 4.9.2).
- PSE-36: 36-Bit page size extension.
  If CPUID.01H:EDX.PSE-36 [bit 17] = 1, the PSE-36 mechanism is supported, indicating that translations using 4-MByte pages with 32-bit paging may produce physical addresses with more than 32 bits (see Section 4.3).
- PCID: process-context identifiers.
  If CPUID.01H:ECX.PCID [bit 17] = 1, CR4.PCIDE may be set to 1, enabling process-context identifiers (see Section 4.10.1).

…

Table 4-4 Format of a 32-Bit Page-Directory Entry that Maps a 4-MByte Page

…

Table 4-6 Format of a 32-Bit Page-Table Entry that Maps a 4-KByte Page

…

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-MByte page referenced by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 4-MByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-MByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-MByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table 4-5)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
12 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0)[1]
(M–20):13	Bits (M–1):32 of physical address of the 4-MByte page referenced by this entry[2]
21:(M–19)	Reserved (must be 0)
31:22	Bits 31:22 of physical address of the 4-MByte page referenced by this entry

NOTES:

1. See Section 4.1.4 for how to determine whether the PAT is supported.
2. If the PSE-36 mechanism is not supported, M is 32, and this row does not apply. If the PSE-36 mechanism is supported, M is the minimum of 40 and MAXPHYADDR (this row does not apply if MAXPHYADDR = 32). See Section 4.1.4 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0)[1]
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
31:12	Physical address of the 4-KByte page referenced by this entry

NOTES:

1. See Section 4.1.4 for how to determine whether the PAT is supported.

Table 4-9 Format of a PAE Page-Directory Entry that Maps a 2-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 2-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte page referenced by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 2-MByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 2-MByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 2-MByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table 4-10)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
12 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0)[1]
20:13	Reserved (must be 0)
(M–1):21	Physical address of the 2-MByte page referenced by this entry
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

NOTES:
1. See Section 4.1.4 for how to determine whether the PAT is supported.

…

Table 4-11 Format of a PAE Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0)[1]
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
(M–1):12	Physical address of the 4-KByte page referenced by this entry
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

NOTES:
1. See Section 4.1.4 for how to determine whether the PAT is supported.

…

4.5 IA-32E PAGING

A logical processor uses IA-32e paging if CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 1. With IA-32e paging, linear address are translated using a hierarchy of in-memory paging structures located using the contents of CR3. IA-32e paging translates 48-bit linear addresses to 52-bit physical addresses.[1] Although 52 bits corresponds to 4 PBytes, linear addresses are limited to 48 bits; at most 256 TBytes of linear-address space may be accessed at any given time.

IA-32e paging uses a hierarchy of paging structures to produce a translation for a linear address. CR3 is used to locate the first paging-structure, the PML4 table. Use of CR3 with IA-32e paging depends on whether process-context identifiers (PCIDs) have been enabled by setting CR4.PCIDE:

- Table Table 4-12 illustrates how CR3 is used with IA-32e paging if CR4.PCIDE = 0.

Table 4-12 Use of CR3 with IA-32e Paging and CR3.PCIDE = 0

Bit Position(s)	Contents
2:0	Ignored
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the PML4 table during linear-address translation (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the PML4 table during linear-address translation (see Section 4.9.2)
11:5	Ignored
M–1:12	Physical address of the 4-KByte aligned PML4 table used for linear-address translation[1]
63:M	Reserved (must be 0)

NOTES:
1. M is an abbreviation for MAXPHYADDR, which is at most 52; see Section 4.1.4.

- Table Table 4-13 illustrates how CR3 is used with IA-32e paging if CR4.PCIDE = 1.

Table 4-13 Use of CR3 with IA-32e Paging and CR3.PCIDE = 1

Bit Position(s)	Contents
11:0	PCID (see Section 4.10.1)[1]
M–1:12	Physical address of the 4-KByte aligned PML4 table used for linear-address translation[2]
63:M	Reserved (must be 0)[3]

NOTES:
1. Section 4.9.2 explains how the processor determines the memory type used to access the PML4 table during linear-address translation with CR4.PCIDE = 1.
2. M is an abbreviation for MAXPHYADDR, which is at most 52; see Section 4.1.4.
3. See Section 4.10.4.1 for use of bit 63 of the source operand of the MOV to CR3 instruction.

After software modifies the value of CR4.PCIDE, the logical processor immediately begins using CR3 as specified for the new value. For example, if software changes CR4.PCIDE from 1 to 0, the current PCID immediately changes from CR3[11:0] to 000H (see also Section 4.10.4.1). In addition, the logical processor subsequently determines

1. If MAXPHYADDR < 52, bits in the range 51:MAXPHYADDR will be 0 in any physical address used by IA-32e paging. (The corresponding bits are reserved in the paging-structure entries.) See Section 4.1.4 for how to determine MAXPHYADDR.

the memory type used to access the PML4 table using CR3.PWT and CR3.PCD, which had been bits 4:3 of the PCID.

…

Table 4-14 Format of an IA-32e PML4 Entry (PML4E) that References a Page-Directory-Pointer Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page-directory-pointer table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 512-GByte region controlled by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 512-GByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Reserved (must be 0)
11:8	Ignored
M–1:12	Physical address of 4-KByte aligned page-directory-pointer table referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 512-GByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

…

Table 4-15 Format of an IA-32e Page-Directory-Pointer-Table Entry (PDPTE) that Maps a 1-GByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 1-GByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 1-GByte page referenced by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 1-GByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 1-GByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 1-GByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page directory; see Table Table 4-16)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
12 (PAT)	Indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 4.9.2)[1]
29:13	Reserved (must be 0)
(M–1):30	Physical address of the 1-GByte page referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 1-GByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

NOTES:
1. The PAT is supported on all processors that support IA-32e paging.

...

Table 4-16 Format of an IA-32e Page-Directory-Pointer-Table Entry (PDPTE) that References a Page Directory

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page directory
1 (R/W)	Read/write; if 0, writes may not be allowed to the 1-GByte region controlled by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 1-GByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 1-GByte page; see Table Table 4-15)
11:8	Ignored
(M–1):12	Physical address of 4-KByte aligned page directory referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 1-GByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

…

Table 4-17 Format of an IA-32e Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 2-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte page referenced by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 2-MByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 2-MByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 2-MByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table Table 4-17)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
12 (PAT)	Indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2)
20:13	Reserved (must be 0)
(M–1):21	Physical address of the 2-MByte page referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

…

Table 4-18 Format of an IA-32e Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte region controlled by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 2-MByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 2-MByte page; see Table 4-17)
11:8	Ignored
(M–1):12	Physical address of 4-KByte aligned page table referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

…

Table 4-19 Format of an IA-32e Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
(M–1):12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

…

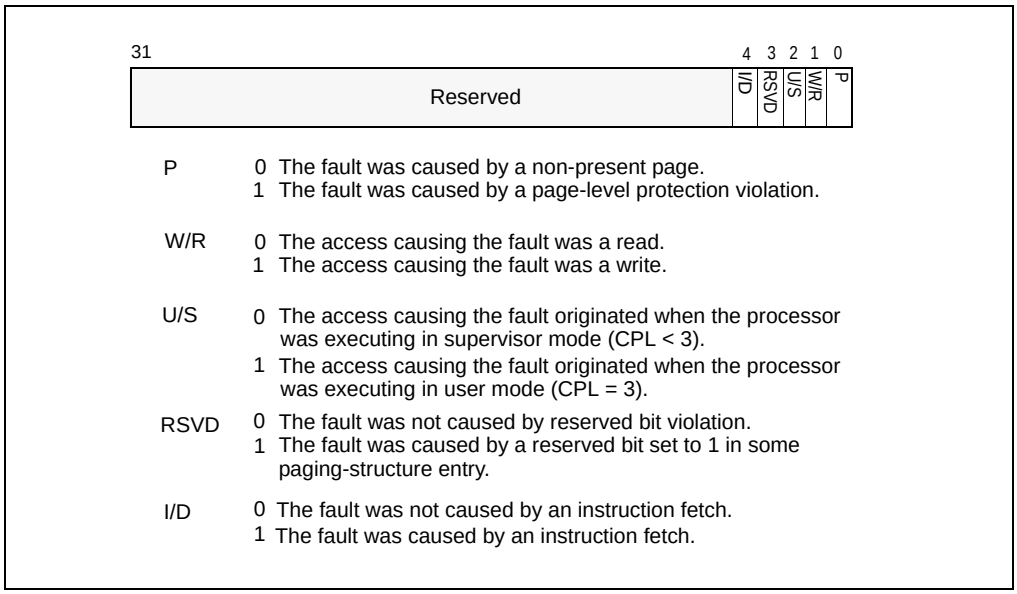

Figure 4-12 Page-Fault Error Code

…

4.9.2 Paging and Memory Typing When the PAT is Supported (Pentium III and More Recent Processor Families)

If the PAT is supported, paging contributes to memory typing in conjunction with the PAT and the memory-type range registers (MTRRs) as specified in Table 11-7 in Section 11.5.2.2.

The PAT is a 64-bit MSR (IA32_PAT; MSR index 277H) comprising eight (8) 8-bit entries (entry *i* comprises bits 8*i*+7:8*i* of the MSR).

For any access to a physical address, the table combines the memory type specified for that physical address by the MTRRs with a memory type selected from the PAT. Table 11-11 in Section 11.12.3 specifies how a memory type is selected from the PAT. Specifically, it comes from entry *i* of the PAT, where *i* is defined as follows:

- For an access to an entry in a paging structure whose address is in CR3 (e.g., the PML4 table with IA-32e paging):
  - For IA-32e paging with CR4.PCIDE = 1, *i* = 0.
  - Otherwise, *i* = 2*PCD+PWT, where the PCD and PWT values come from CR3.
- For an access to a PDE with PAE paging, *i* = 2*PCD+PWT, where the PCD and PWT values come from the relevant PDPTE register.
- For an access to a paging-structure entry X whose address is in another paging-structure entry Y, *i* = 2*PCD+PWT, where the PCD and PWT values come from Y.
- For an access to the physical address that is the translation of a linear address, *i* = 4*PAT+2*PCD+PWT, where the PAT, PCD, and PWT values come from the relevant PTE (if the translation uses a 4-KByte page), the relevant PDE (if the translation uses

a 2-MByte page or a 4-MByte page), or the relevant PDPTE (if the translation uses a 1-GByte page).

...

4.10 CACHING TRANSLATION INFORMATION

The Intel-64 and IA-32 architectures may accelerate the address-translation process by caching data from the paging structures on the processor. Because the processor does not ensure that the data that it caches are always consistent with the structures in memory, it is important for software developers to understand how and when the processor may cache such data. They should also understand what actions software can take to remove cached data that may be inconsistent and when it should do so. This section provides software developers information about the relevant processor operation.

Section 4.10.1 introduces process-context identifiers (PCIDs), which a logical processor may use to distinguish information cached for different linear-address spaces. Section 4.10.2 and Section 4.10.3 describe how the processor may cache information in translation lookaside buffers (TLBs) and paging-structure caches, respectively. Section 4.10.4 explains how software can remove inconsistent cached information by invalidating portions of the TLBs and paging-structure caches. Section 4.10.5 describes special considerations for multiprocessor systems.

4.10.1 Process-Context Identifiers (PCIDs)

Process-context identifiers (**PCIDs**) are a facility by which a logical processor may cache information for multiple linear-address spaces. The processor may retain cached information when software switches to a different linear-address space with a different PCID (e.g., by loading CR3; see Section 4.10.4.1 for details).

A PCID is a 12-bit identifier. Non-zero PCIDs are enabled by setting the PCIDE flag (bit 17) of CR4. If CR4.PCIDE = 0, the current PCID is always 000H; otherwise, the current PCID is the value of bits 11:0 of CR3. Not all processors allow CR4.PCIDE to be set to 1; see Section 4.1.4 for how to determine whether this is allowed.

The processor ensures that CR4.PCIDE can be 1 only in IA-32e mode (thus, 32-bit paging and PAE paging use only PCID 000H). In addition, software can change CR4.PCIDE from 0 to 1 only if CR3[11:0] = 000H. These requirements are enforced by the following limitations on the MOV CR instruction:

- MOV to CR4 causes a general-protection exception (#GP) if it would change CR4.PCIDE from 0 to 1 and either IA32_EFER.LMA = 0 or CR3[11:0] ≠ 000H.
- MOV to CR0 causes a general-protection exception if it would clear CR0.PG to 0 while CR4.PCIDE = 1.

When a logical processor creates entries in the TLBs (Section 4.10.2) and paging-structure caches (Section 4.10.3), it associates those entries with the current PCID. When using entries in the TLBs and paging-structure caches to translate a linear address, a logical processor uses only those entries associated with the current PCID (see Section 4.10.2.4 for an exception).

If CR4.PCIDE = 0, a logical processor does not cache information for any PCID other than 000H. This is because (1) if CR4.PCIDE = 0, the logical processor will associate any

newly cached information with the current PCID, 000H; and (2) if MOV to CR4 clears CR4.PCIDE, all cached information is invalidated (see Section 4.10.4.1).

NOTE

In revisions of this manual that were produced when no processors allowed CR4.PCIDE to be set to 1, Section 4.10 discussed the caching of translation information without any reference to PCIDs. While the section now refers to PCIDs in its specification of this caching, this documentation change is not intended to imply any change to the behavior of processors that do not allow CR4.PCIDE to be set to 1.

...

4.10.2.2 Caching Translations in TLBs

The processor may accelerate the paging process by caching individual translations in **translation lookaside buffers** (**TLBs**). Each entry in a TLB is an individual translation. Each translation is referenced by a page number. It contains the following information from the paging-structure entries used to translate linear addresses with the page number:

- The physical address corresponding to the page number (the page frame).
- The access rights from the paging-structure entries used to translate linear addresses with the page number (see Section 4.6):
  - The logical-AND of the R/W flags.
  - The logical-AND of the U/S flags.
  - The logical-OR of the XD flags (necessary only if IA32_EFER.NXE = 1).
- Attributes from a paging-structure entry that identifies the final page frame for the page number (either a PTE or a paging-structure entry in which the PS flag is 1):
  - The dirty flag (see Section 4.8).
  - The memory type (see Section 4.9).

(TLB entries may contain other information as well. A processor may implement multiple TLBs, and some of these may be for special purposes, e.g., only for instruction fetches. Such special-purpose TLBs may not contain some of this information if it is not necessary. For example, a TLB used only for instruction fetches need not contain information about the R/W and dirty flags.)

As noted in Section 4.10.1, any TLB entries created by a logical processor are associated with the current PCID.

Processors need not implement any TLBs. Processors that do implement TLBs may invalidate any TLB entry at any time. Software should not rely on the existence of TLBs or on the retention of TLB entries.

4.10.2.3 Details of TLB Use

Because the TLBs cache only valid translations, there can be a TLB entry for a page number only if the P flag is 1 and the reserved bits are 0 in each of the paging-structure entries used to translate that page number. In addition, the processor does not cache a translation for a page number unless the accessed flag is 1 in each of the paging-structure entries used during translation; before caching a translation, the processor sets any of these accessed flags that is not already 1.

The processor may cache translations required for prefetches and for accesses that are a result of speculative execution that would never actually occur in the executed code path.

If the page number of a linear address corresponds to a TLB entry associated with the current PCID, the processor may use that TLB entry to determine the page frame, access rights, and other attributes for accesses to that linear address. In this case, the processor may not actually consult the paging structures in memory. The processor may retain a TLB entry unmodified even if software subsequently modifies the relevant paging-structure entries in memory. See Section 4.10.4.2 for how software can ensure that the processor uses the modified paging-structure entries.

If the paging structures specify a translation using a page larger than 4 KBytes, some processors may choose to cache multiple smaller-page TLB entries for that translation. Each such TLB entry would be associated with a page number corresponding to the smaller page size (e.g., bits 47:12 of a linear address with IA-32e paging), even though part of that page number (e.g., bits 20:12) are part of the offset with respect to the page specified by the paging structures. The upper bits of the physical address in such a TLB entry are derived from the physical address in the PDE used to create the translation, while the lower bits come from the linear address of the access for which the translation is created. There is no way for software to be aware that multiple translations for smaller pages have been used for a large page.

If software modifies the paging structures so that the page size used for a 4-KByte range of linear addresses changes, the TLBs may subsequently contain multiple translations for the address range (one for each page size). A reference to a linear address in the address range may use any of these translations. Which translation is used may vary from one execution to another, and the choice may be implementation-specific.

4.10.2.4 Global Pages

The Intel-64 and IA-32 architectures also allow for **global pages** when the PGE flag (bit 7) is 1 in CR4. If the G flag (bit 8) is 1 in a paging-structure entry that maps a page (either a PTE or a paging-structure entry in which the PS flag is 1), any TLB entry cached for a linear address using that paging-structure entry is considered to be **global**. Because the G flag is used only in paging-structure entries that map a page, and because information from such entries are not cached in the paging-structure caches, the global-page feature does not affect the behavior of the paging-structure caches.

A logical processor may use a global TLB entry to translate a linear address, even if the TLB entry is associated with a PCID different from the current PCID.

...

4.10.3.1 Caches for Paging Structures

A processor may support any or of all the following paging-structure caches:

- **PML4 cache** (IA-32e paging only). Each PML4-cache entry is referenced by a 9-bit value and is used for linear addresses for which bits 47:39 have that value. The entry contains information from the PML4E used to translate such linear addresses:
  - The physical address from the PML4E (the address of the page-directory-pointer table).
  - The value of the R/W flag of the PML4E.
  - The value of the U/S flag of the PML4E.

— The value of the XD flag of the PML4E.

— The values of the PCD and PWT flags of the PML4E.

The following items detail how a processor may use the PML4 cache:

— If the processor has a PML4-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML4E in memory).

— The processor does not create a PML4-cache entry unless the P flag is 1 and all reserved bits are 0 in the PML4E in memory.

— The processor does not create a PML4-cache entry unless the accessed flag is 1 in the PML4E in memory; before caching a translation, the processor sets the accessed flag if it is not already 1.

— The processor may create a PML4-cache entry even if there are no translations for any linear address that might use that entry (e.g., because the P flags are 0 in all entries in the referenced page-directory-pointer table).

— If the processor creates a PML4-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML4E in memory.

- **PDPTE cache** (IA-32e paging only).[1] Each PDPTE-cache entry is referenced by an 18-bit value and is used for linear addresses for which bits 47:30 have that value. The entry contains information from the PML4E and PDPTE used to translate such linear addresses:

  — The physical address from the PDPTE (the address of the page directory). (No PDPTE-cache entry is created for a PDPTE that maps a 1-GByte page.)

  — The logical-AND of the R/W flags in the PML4E and the PDPTE.

  — The logical-AND of the U/S flags in the PML4E and the PDPTE.

  — The logical-OR of the XD flags in the PML4E and the PDPTE.

  — The values of the PCD and PWT flags of the PDPTE.

  The following items detail how a processor may use the PDPTE cache:

  — If the processor has a PDPTE-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML4E and the PDPTE in memory).

  — The processor does not create a PDPTE-cache entry unless the P flag is 1, the PS flag is 0, and the reserved bits are 0 in the PML4E and the PDPTE in memory.

  — The processor does not create a PDPTE-cache entry unless the accessed flags are 1 in the PML4E and the PDPTE in memory; before caching a translation, the processor sets any accessed flags that are not already 1.

  — The processor may create a PDPTE-cache entry even if there are no translations for any linear address that might use that entry.

  — If the processor creates a PDPTE-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML4E or PDPTE in memory.

- **PDE cache**. The use of the PDE cache depends on the paging mode:

1. With PAE paging, the PDPTEs are stored in internal, non-architectural registers. The operation of these registers is described in Section 4.4.1 and differs from that described here.

— For 32-bit paging, each PDE-cache entry is referenced by a 10-bit value and is used for linear addresses for which bits 31:22 have that value.

— For PAE paging, each PDE-cache entry is referenced by an 11-bit value and is used for linear addresses for which bits 31:21 have that value.

— For IA-32e paging, each PDE-cache entry is referenced by a 27-bit value and is used for linear addresses for which bits 47:21 have that value.

A PDE-cache entry contains information from the PML4E, PDPTE, and PDE used to translate the relevant linear addresses (for 32-bit paging and PAE paging, only the PDE applies):

— The physical address from the PDE (the address of the page table). (No PDE-cache entry is created for a PDE that maps a page.)

— The logical-AND of the R/W flags in the PML4E, PDPTE, and PDE.

— The logical-AND of the U/S flags in the PML4E, PDPTE, and PDE.

— The logical-OR of the XD flags in the PML4E, PDPTE, and PDE.

— The values of the PCD and PWT flags of the PDE.

The following items detail how a processor may use the PDE cache (references below to PML4Es and PDPTEs apply on to IA-32e paging):

— If the processor has a PDE-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML4E, the PDPTE, and the PDE in memory).

— The processor does not create a PDE-cache entry unless the P flag is 1, the PS flag is 0, and the reserved bits are 0 in the PML4E, the PDPTE, and the PDE in memory.

— The processor does not create a PDE-cache entry unless the accessed flag is 1 in the PML4E, the PDPTE, and the PDE in memory; before caching a translation, the processor sets any accessed flags that are not already 1.

— The processor may create a PDE-cache entry even if there are no translations for any linear address that might use that entry.

— If the processor creates a PDE-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML4E, the PDPTE, or the PDE in memory.

Information from a paging-structure entry can be included in entries in the paging-structure caches for other paging-structure entries referenced by the original entry. For example, if the R/W flag is 0 in a PML4E, then the R/W flag will be 0 in any PDPTE-cache entry for a PDPTE from the page-directory-pointer table referenced by that PML4E. This is because the R/W flag of each such PDPTE-cache entry is the logical-AND of the R/W flags in the appropriate PML4E and PDPTE.

The paging-structure caches contain information only from paging-structure entries that reference other paging structures (and not those that map pages). Because the G flag is not used in such paging-structure entries, the global-page feature does not affect the behavior of the paging-structure caches.

The processor may create entries in paging-structure caches for translations required for prefetches and for accesses that are a result of speculative execution that would never actually occur in the executed code path.

As noted in Section 4.10.1, any entries created in paging-structure caches by a logical processor are associated with the current PCID.

A processor may or may not implement any of the paging-structure caches. Software should rely on neither their presence nor their absence. The processor may invalidate entries in these caches at any time. Because the processor may create the cache entries at the time of translation and not update them following subsequent modifications to the paging structures in memory, software should take care to invalidate the cache entries appropriately when causing such modifications. The invalidation of TLBs and the paging-structure caches is described in Section 4.10.4.

4.10.3.2 Using the Paging-Structure Caches to Translate Linear Addresses

When a linear address is accessed, the processor uses a procedure such as the following to determine the physical address to which it translates and whether the access should be allowed:

- If the processor finds a TLB entry that is for the page number of the linear address and that is associated with the current PCID (or which is global), it may use the physical address, access rights, and other attributes from that entry.
- If the processor does not find a relevant TLB entry, it may use the upper bits of the linear address to select an entry from the PDE cache that is associated with the current PCID (Section 4.10.3.1 indicates which bits are used in each paging mode). It can then use that entry to complete the translation process (locating a PTE, etc.) as if it had traversed the PDE (and, for IA-32e paging, the PDPTE and PML4) corresponding to the PDE-cache entry.
- The following items apply when IA-32e paging is used:
  - If the processor does not find a relevant TLB entry or a relevant PDE-cache entry, it may use bits 47:30 of the linear address to select an entry from the PDPTE cache that is associated with the current PCID. It can then use that entry to complete the translation process (locating a PDE, etc.) as if it had traversed the PDPTE and the PML4 corresponding to the PDPTE-cache entry.
  - If the processor does not find a relevant TLB entry, a relevant PDE-cache entry, or a relevant PDPTE-cache entry, it may use bits 47:39 of the linear address to select an entry from the PML4 cache that is associated with the current PCID. It can then use that entry to complete the translation process (locating a PDPTE, etc.) as if it had traversed the corresponding PML4.

(Any of the above steps would be skipped if the processor does not support the cache in question.)

If the processor does not find a TLB or paging-structure-cache entry for the linear address, it uses the linear address to traverse the entire paging-structure hierarchy, as described in Section 4.3, Section 4.4.2, and Section 4.5.

4.10.3.3 Multiple Cached Entries for a Single Paging-Structure Entry

The paging-structure caches and TLBs and paging-structure caches may contain multiple entries associated with a single PCID and with information derived from a single paging-structure entry. The following items give some examples for IA-32e paging:

- Suppose that two PML4Es contain the same physical address and thus reference the same page-directory-pointer table. Any PDPTE in that table may result in two PDPTE-cache entries, each associated with a different set of linear addresses. Specifically, suppose that the n_1^{th} and n_2^{th} entries in the PML4 table contain the same physical address. This implies that the physical address in the m^{th} PDPTE in the page-directory-pointer table would appear in the PDPTE-cache entries associated with both p_1 and p_2, where $(p_1 \gg 9) = n_1$, $(p_2 \gg 9) = n_2$, and $(p_1 \& 1FFH) = (p_2 \& 1FFH) =$

m. This is because both PDPTE-cache entries use the same PDPTE, one resulting from a reference from the n_1^{th} PML4E and one from the n_2^{th} PML4E.

- Suppose that the first PML4E (i.e., the one in position 0) contains the physical address X in CR3 (the physical address of the PML4 table). This implies the following:
  - Any PML4-cache entry associated with linear addresses with 0 in bits 47:39 contains address X.
  - Any PDPTE-cache entry associated with linear addresses with 0 in bits 47:30 contains address X. This is because the translation for a linear address for which the value of bits 47:30 is 0 uses the value of bits 47:39 (0) to locate a page-directory-pointer table at address X (the address of the PML4 table). It then uses the value of bits 38:30 (also 0) to find address X again and to store that address in the PDPTE-cache entry.
  - Any PDE-cache entry associated with linear addresses with 0 in bits 47:21 contains address X for similar reasons.
  - Any TLB entry for page number 0 (associated with linear addresses with 0 in bits 47:12) translates to page frame X » 12 for similar reasons.

  The same PML4E contributes its address X to all these cache entries because the self-referencing nature of the entry causes it to be used as a PML4E, a PDPTE, a PDE, and a PTE.

4.10.4 Invalidation of TLBs and Paging-Structure Caches

As noted in Section 4.10.2 and Section 4.10.3, the processor may create entries in the TLBs and the paging-structure caches when linear addresses are translated, and it may retain these entries even after the paging structures used to create them have been modified. To ensure that linear-address translation uses the modified paging structures, software should take action to invalidate any cached entries that may contain information that has since been modified.

4.10.4.1 Operations that Invalidate TLBs and Paging-Structure Caches

The following instructions invalidate entries in the TLBs and the paging-structure caches:

- INVLPG. This instruction takes a single operand, which is a linear address. The instruction invalidates any TLB entries that are for a page number corresponding to the linear address and that are associated with the current PCID. It also invalidates any global TLB entries with that page number, regardless of PCID (see Section 4.10.2.4).[1] INVLPG also invalidates all entries in all paging-structure caches associated with the current PCID, regardless of the linear addresses to which they correspond.
- MOV to CR3. The behavior of the instruction depends on the value of CR4.PCIDE:
  - If CR4.PCIDE = 0, the instruction invalidates all TLB entries associated with PCID 000H except those for global pages. It also invalidates all entries in all paging-structure caches associated with PCID 000H.
  - If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 0, the instruction invalidates all TLB entries associated with the PCID specified in bits 11:0 of the instruction's source operand except those for global pages. It

1. If the paging structures map the linear address using a page larger than 4 KBytes and there are multiple TLB entries for that page (see Section 4.10.2.3), the instruction invalidates all of them.

also invalidates all entries in all paging-structure caches associated with that PCID. It is not required to invalidate entries in the TLBs and paging-structure caches that are associated with other PCIDs.

— If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 1, the instruction is not required to invalidate any TLB entries or entries in paging-structure caches.

- MOV to CR4. The instruction invalidates all TLB entries (including global entries) and all entries in all paging-structure caches (for all PCIDs) if either (1) it changes the value of the CR4.PGE flag;[1] or (2) it changes the value of the CR4.PCIDE from 1 to 0.
- Task switch. If a task switch changes the value of CR3, it invalidates all TLB entries associated with PCID 000H except those for global pages. It also invalidates all entries in all paging-structure caches for associated with PCID 000H.[2]
- VMX transitions. See Section 4.11.1.

The processor is always free to invalidate additional entries in the TLBs and paging-structure caches. The following are some examples:

- INVLPG may invalidate TLB entries for pages other than the one corresponding to its linear-address operand. It may invalidate TLB entries and paging-structure-cache entries associated with PCIDs other than the current PCID.
- MOV to CR3 may invalidate TLB entries for global pages. If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 0, it may invalidate TLB entries and entries in the paging-structure caches associated with PCIDs other than the current PCID. It may invalidate entries if CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 1.
- On a processor supporting Hyper-Threading Technology, invalidations performed on one logical processor may invalidate entries in the TLBs and paging-structure caches used by other logical processors.

(Other instructions and operations may invalidate entries in the TLBs and the paging-structure caches, but the instructions identified above are recommended.)

In addition to the instructions identified above, page faults invalidate entries in the TLBs and paging-structure caches. In particular, a page-fault exception resulting from an attempt to use a linear address will invalidate any TLB entries that are for a page number corresponding to that linear address and that are associated with the current PCID. it also invalidates all entries in the paging-structure caches that would be used for that linear address and that are associated with the current PCID.[3] These invalidations ensure that the page-fault exception will not recur (if the faulting instruction is re-executed) if it would not be caused by the contents of the paging structures in memory (and if, therefore, it resulted from cached entries that were not invalidated after the paging structures were modified in memory).

As noted in Section 4.10.2, some processors may choose to cache multiple smaller-page TLB entries for a translation specified by the paging structures to use a page larger than 4 KBytes. There is no way for software to be aware that multiple translations for smaller pages have been used for a large page. The INVLPG instruction and page faults provide

1. If CR4.PGE is changing from 0 to 1, there were no global TLB entries before the execution; if CR4.PGE is changing from 1 to 0, there will be no global TLB entries after the execution.
2. Task switches do not occur in IA-32e mode and thus cannot occur with IA-32e paging. Since CR4.PCIDE can be set only with IA-32e paging, task switches occur only with CR4.PCIDE = 0.
3. Unlike INVLPG, page faults need not invalidate **all** entries in the paging-structure caches, only those that would be used to translate the faulting linear address.

the same assurances that they provide when a single TLB entry is used: they invalidate all TLB entries corresponding to the translation specified by the paging structures.

4.10.4.2 Recommended Invalidation

The following items provide some recommendations regarding when software should perform invalidations:

- If software modifies a paging-structure entry that identifies the final page frame for a page number (either a PTE or a paging-structure entry in which the PS flag is 1), it should execute INVLPG for any linear address with a page number whose translation uses that PTE.[1] (If the paging-structure entry may be used in the translation of different page numbers — see Section 4.10.3.3 — software should execute INVLPG for linear addresses with each of those page numbers; alternatively, it could use MOV to CR3 or MOV to CR4.)
- If software modifies a paging-structure entry that references another paging structure, it may use one of the following approaches depending upon the types and number of translations controlled by the modified entry:
  - — Execute INVLPG for linear addresses with each of the page numbers with translations that would use the entry. However, if no page numbers that would use the entry have translations (e.g., because the P flags are 0 in all entries in the paging structure referenced by the modified entry), it remains necessary to execute INVLPG at least once.
  - — Execute MOV to CR3 if the modified entry controls no global pages.
  - — Execute MOV to CR4 to modify CR4.PGE.
- If CR4.PCIDE = 1 and software modifies a paging-structure entry that does not map a page or in which the G flag (bit 8) is 0, additional steps are required if the entry may be used for PCIDs other than the current one. Any one of the following suffices:
  - — Execute MOV to CR4 to modify CR4.PGE, either immediately or before again using any of the affected PCIDs. For example, software could use different (previously unused) PCIDs for the processes that used the affected PCIDs.
  - — For each affected PCID, execute MOV to CR3 to make that PCID current (and to load the address of the appropriate PML4 table). If the modified entry controls no global pages and bit 63 of the source operand to MOV to CR3 was 0, no further steps are required. Otherwise, execute INVLPG for linear addresses with each of the page numbers with translations that would use the entry; if no page numbers that would use the entry have translations, execute INVLPG at least once.
- If software using PAE paging modifies a PDPTE, it should reload CR3 with the register's current value to ensure that the modified PDPTE is loaded into the corresponding PDPTE register (see Section 4.4.1).
- If the nature of the paging structures is such that a single entry may be used for multiple purposes (see Section 4.10.3.3), software should perform invalidations for all of these purposes. For example, if a single entry might serve as both a PDE and PTE, it may be necessary to execute INVLPG with two (or more) linear addresses, one that uses the entry as a PDE and one that uses it as a PTE. (Alternatively, software could use MOV to CR3 or MOV to CR4.)
- As noted in Section 4.10.2, the TLBs may subsequently contain multiple translations for the address range if software modifies the paging structures so that the page size

1. One execution of INVLPG is sufficient even for a page with size greater than 4 KBytes.

used for a 4-KByte range of linear addresses changes. A reference to a linear address in the address range may use any of these translations.

Software wishing to prevent this uncertainty should not write to a paging-structure entry in a way that would change, for any linear address, both the page size and either the page frame, access rights, or other attributes. It can instead use the following algorithm: first clear the P flag in the relevant paging-structure entry (e.g., PDE); then invalidate any translations for the affected linear addresses (see Section 4.10.4.2); and then modify the relevant paging-structure entry to set the P flag and establish modified translation(s) for the new page size.

- Software should clear bit 63 of the source operand to a MOV to CR3 instruction that establishes a PCID that had been used earlier for a different linear-address space (e.g., with a different value in bits 51:12 of CR3). This ensures invalidation of any information that may have been cached for the previous linear-address space.

  This assumes that both linear-address spaces use the same global pages and that it is thus not necessary to invalidate any global TLB entries. If that is not the case, software should invalidate those entries by executing MOV to CR4 to modify CR4.PGE.

...

4.11.2 VMX Support for Address Translation

Chapter 25, “VMX Support for Address Translation,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B* describe two features of the virtual-machine extensions (VMX) that interact directly with paging. These are **virtual-processor identifiers** (**VPIDs**) and the **extended page table** mechanism (**EPT**).

VPIDs provide a way for software to identify to the processor the address spaces for different “virtual processors.” The processor may use this identification to maintain concurrently information for multiple address spaces in its TLBs and paging-structure caches, even when non-zero PCIDs are not being used. See Section 25.1 for details.

...

12. Updates to Chapter 6, Volume 3A

Change bars show changes to Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A:* System Programming Guide, Part 1.

...

6.2 EXCEPTION AND INTERRUPT VECTORS

To aid in handling exceptions and interrupts, each architecturally defined exception and each interrupt condition requiring special handling by the processor is assigned a unique identification number, called a vector number. The processor uses the vector number assigned to an exception or interrupt as an index into the interrupt descriptor table (IDT). The table provides the entry point to an exception or interrupt handler (see Section 6.10, “Interrupt Descriptor Table (IDT)”).

The allowable range for vector numbers is 0 to 255. Vector numbers in the range 0 through 31 are reserved by the Intel 64 and IA-32 architectures for architecture-defined exceptions and interrupts. Not all of the vector numbers in this range have a currently defined function. The unassigned vector numbers in this range are reserved. Do not use the reserved vector numbers.

Vector numbers in the range 32 to 255 are designated as user-defined interrupts and are not reserved by the Intel 64 and IA-32 architecture. These interrupts are generally assigned to external I/O devices to enable those devices to send interrupts to the processor through one of the external hardware interrupt mechanisms (see Section 6.3, “Sources of Interrupts”).

Table 6-1 shows vector number assignments for architecturally defined exceptions and for the NMI interrupt. This table gives the exception type (see Section 6.5, “Exception Classifications”) and indicates whether an error code is saved on the stack for the exception. The source of each predefined exception and the NMI interrupt is also given.

…

13. Updates to Chapter 10, Volume 3A

Change bars show changes to Chapter 10 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A:* System Programming Guide, Part 1.

…

10.5.4 APIC Timer

The local APIC unit contains a 32-bit programmable timer that is available to software to time events or operations. This timer is set up by programming four registers: the divide configuration register (see Figure Figure 10-10), the initial-count and current-count registers (see Figure Figure 10-11), and the LVT timer register (see Figure 10-8).

If CPUID.06H:EAX.ARAT[bit 2] = 1, the processor’s APIC timer runs at a constant rate regardless of P-state transitions and it continues to run at the same rate in deep C-states.

If CPUID.06H:EAX.ARAT[bit 2] = 0 or if CPUID 06H is not supported, the APIC timer may temporarily stop while the processor is in deep C-states or during transitions caused by Enhanced Intel SpeedStep® Technology.

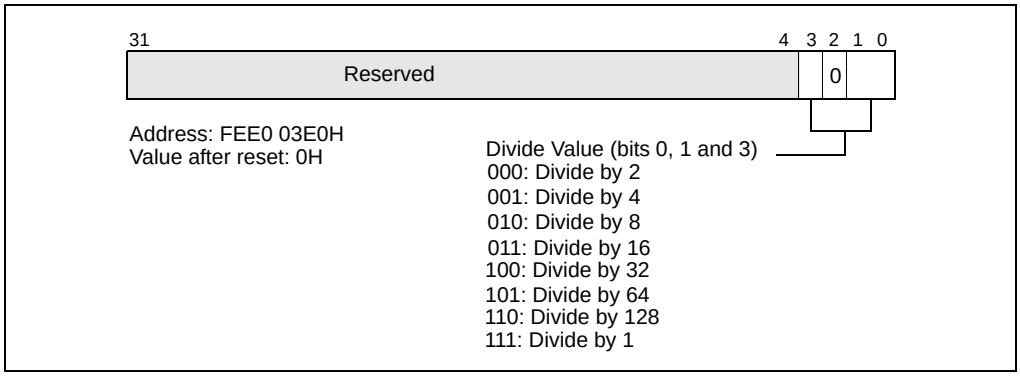

Figure 10-10 Divide Configuration Register

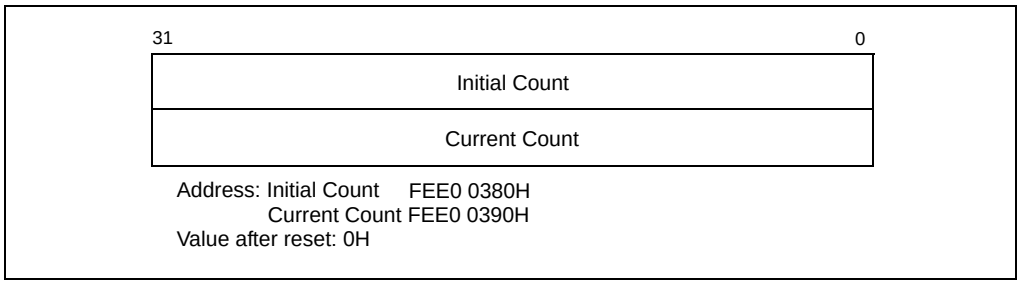

Figure 10-11 Initial Count and Current Count Registers

The time base for the timer is derived from the processor's bus clock, divided by the value specified in the divide configuration register.

The timer can be configured through the timer LVT entry for one-shot or periodic operation. In one-shot mode, the timer is started by programming its initial-count register. The initial count value is then copied into the current-count register and count-down begins. After the timer reaches zero, an timer interrupt is generated and the timer remains at its 0 value until reprogrammed.

In periodic mode, the current-count register is automatically reloaded from the initial-count register when the count reaches 0 and a timer interrupt is generated, and the count-down is repeated. If during the count-down process the initial-count register is set, counting will restart, using the new initial-count value. The initial-count register is a read-write register; the current-count register is read only.

A write of 0 to the initial-count register effectively stops the local APIC timer, in both one-shot and periodic mode.

The LVT timer register determines the vector number that is delivered to the processor with the timer interrupt that is generated when the timer count reaches zero. The mask flag in the LVT timer register can be used to mask the timer interrupt.

…

10.6.1 Interrupt Command Register (ICR)

The interrupt command register (ICR) is a 64-bit[1] local APIC register (see Figure 10-12) that allows software running on the processor to specify and send interprocessor interrupts (IPIs) to other processors in the system.

…

10.6.2.2 Logical Destination Mode

…

The interpretation of MDA for the two models is described in the following paragraphs.

1. **Flat Model** — This model is selected by programming DFR bits 28 through 31 to 1111. Here, a unique logical APIC ID can be established for up to 8 local APICs by

1. In XAPIC mode the ICR is addressed as two 32-bit registers, ICR_LOW (FFE0 0300H) and ICR_HIGH (FFE0 0310H).

setting a different bit in the logical APIC ID field of the LDR for each local APIC. A group of local APICs can then be selected by setting one or more bits in the MDA.

Each local APIC performs a bit-wise AND of the MDA and its logical APIC ID. If a true condition is detected, the local APIC accepts the IPI message. A broadcast to all APICs is achieved by setting the MDA to 1s.

2. **Cluster Model** — This model is selected by programming DFR bits 28 through 31 to 0000. This model supports two basic destination schemes: flat cluster and hierarchical cluster.

   The flat cluster destination model is only supported for P6 family and Pentium processors. Using this model, all APICs are assumed to be connected through the APIC bus. Bits 60 through 63 of the MDA contains the encoded address of the destination cluster and bits 56 through 59 identify up to four local APICs within the cluster (each bit is assigned to one local APIC in the cluster, as in the flat connection model). To identify one or more local APICs, bits 60 through 63 of the MDA are compared with bits 28 through 31 of the LDR to determine if a local APIC is part of the cluster. Bits 56 through 59 of the MDA are compared with Bits 24 through 27 of the LDR to identify a local APICs within the cluster.

   Sets of processors within a cluster can be specified by writing the target cluster address in bits 60 through 63 of the MDA and setting selected bits in bits 56 through 59 of the MDA, corresponding to the chosen members of the cluster. In this mode, 15 clusters (with cluster addresses of 0 through 14) each having 4 local APICs can be specified in the message. For the P6 and Pentium processor's local APICs, however, the APIC arbitration ID supports only 15 APIC agents. Therefore, the total number of processors and their local APICs supported in this mode is limited to 15. Broadcast to all local APICs is achieved by setting all destination bits to one. This guarantees a match on all clusters and selects all APICs in each cluster. A broadcast IPI or I/O subsystem broadcast interrupt with lowest priority delivery mode is not supported in cluster mode and must not be configured by software.

   The hierarchical cluster destination model can be used with Pentium 4, Intel Xeon, P6 family, or Pentium processors. With this model, a hierarchical network can be created by connecting different flat clusters via independent system or APIC buses. This scheme requires a cluster manager within each cluster, which is responsible for handling message passing between system or APIC buses. One cluster contains up to 4 agents. Thus 15 cluster managers, each with 4 agents, can form a network of up to 60 APIC agents. Note that hierarchical APIC networks requires a special cluster manager device, which is not part of the local or the I/O APIC units.

...

14. Updates to Chapter 11, Volume 3A

Change bars show changes to Chapter 11 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

Table 11-1 Characteristics of the Caches, TLBs, Store Buffer, and

Write Combining Buffer in Intel 64 and IA-32 Processors

Cache or Buffer	Characteristics
Trace Cache[1]	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 12 Kμops, 8-way set associative. ▪ Intel Core i7, Intel Core 2 Duo, Intel Atom, Intel Core Duo, Intel Core Solo, Pentium M processor: not implemented. ▪ P6 family and Pentium processors: not implemented.
L1 Instruction Cache	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): not implemented. ▪ Intel Core i7 processor: 32-KByte, 4-way set associative. ▪ Intel Core 2 Duo, Intel Atom, Intel Core Duo, Intel Core Solo, Pentium M processor: 32-KByte, 8-way set associative. ▪ P6 family and Pentium processors: 8- or 16-KByte, 4-way set associative, 32-byte cache line size; 2-way set associative for earlier Pentium processors.
L1 Data Cache	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 8-KByte, 4-way set associative, 64-byte cache line size. ▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 16-KByte, 8-way set associative, 64-byte cache line size. ▪ Intel Atom processors: 24-KByte, 6-way set associative, 64-byte cache line size. ▪ Intel Core i7, Intel Core 2 Duo, Intel Core Duo, Intel Core Solo, Pentium M and Intel Xeon processors: 32-KByte, 8-way set associative, 64-byte cache line size. ▪ P6 family processors: 16-KByte, 4-way set associative, 32-byte cache line size; 8-KBytes, 2-way set associative for earlier P6 family processors. ▪ Pentium processors: 16-KByte, 4-way set associative, 32-byte cache line size; 8-KByte, 2-way set associative for earlier Pentium processors.
L2 Unified Cache	▪ Intel Core 2 Duo and Intel Xeon processors: up to 4-MByte (or 4MBx2 in quadcore processors), 16-way set associative, 64-byte cache line size. ▪ Intel Core 2 Duo and Intel Xeon processors: up to 6-MByte (or 6MBx2 in quadcore processors), 24-way set associative, 64-byte cache line size. ▪ Intel Core i7, i5, i3 processors: 256KBbyte, 8-way set associative, 64-byte cache line size. ▪ Intel Atom processors: 512-KByte, 8-way set associative, 64-byte cache line size. ▪ Intel Core Duo, Intel Core Solo processors: 2-MByte, 8-way set associative, 64-byte cache line size ▪ Pentium 4 and Intel Xeon processors: 256, 512, 1024, or 2048-KByte, 8-way set associative, 64-byte cache line size, 128-byte sector size. ▪ Pentium M processor: 1 or 2-MByte, 8-way set associative, 64-byte cache line size. ▪ P6 family processors: 128-KByte, 256-KByte, 512-KByte, 1-MByte, or 2-MByte, 4-way set associative, 32-byte cache line size. ▪ Pentium processor (external optional): System specific, typically 256- or 512-KByte, 4-way set associative, 32-byte cache line size.

Cache or Buffer	Characteristics
L3 Unified Cache	▪ Intel Xeon processors: 512-KByte, 1-MByte, 2-MByte, or 4-MByte, 8-way set associative, 64-byte cache line size, 128-byte sector size. ▪ Intel Core i7 processor, Intel Xeon processor 5500: Up to 8MByte, 16-way set associative, 64-byte cache line size. ▪ Intel Xeon processor 5600: Up to 12MByte, 64-byte cache line size. ▪ Intel Xeon processor 7500: Up to 24MByte, 64-byte cache line size.
Instruction TLB (4-KByte Pages)	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 128 entries, 4-way set associative. ▪ Intel Atom processors: 32-entries, fully associative. ▪ Intel Core i7, i5, i3 processors: 64-entries per thread (128-entries per core), 4-way set associative. ▪ Intel Core 2 Duo, Intel Core Duo, Intel Core Solo processors, Pentium M processor: 128 entries, 4-way set associative. ▪ P6 family processors: 32 entries, 4-way set associative. ▪ Pentium processor: 32 entries, 4-way set associative; fully set associative for Pentium processors with MMX technology.
Data TLB (4-KByte Pages)	▪ Intel Core i7, i5, i3 processors, DTLB0: 64-entries, 4-way set associative. ▪ Intel Core 2 Duo processors: DTLB0, 16 entries, DTLB1, 256 entries, 4 ways. ▪ Intel Atom processors: 16-entry-per-thread micro-TLB, fully associative; 64-entry DTLB, 4-way set associative; 16-entry PDE cache, fully associative. ▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 64 entry, fully set associative, shared with large page DTLB. ▪ Intel Core Duo, Intel Core Solo processors, Pentium M processor: 128 entries, 4-way set associative. ▪ Pentium and P6 family processors: 64 entries, 4-way set associative; fully set, associative for Pentium processors with MMX technology.
Instruction TLB (Large Pages)	▪ Intel Core i7, i5, i3 processors: 7-entries per thread, fully associative. ▪ Intel Core 2 Duo processors: 4 entries, 4 ways. ▪ Pentium 4 and Intel Xeon processors: large pages are fragmented. ▪ Intel Core Duo, Intel Core Solo, Pentium M processor: 2 entries, fully associative. ▪ P6 family processors: 2 entries, fully associative. ▪ Pentium processor: Uses same TLB as used for 4-KByte pages.
Data TLB (Large Pages)	▪ Intel Core i7, i5, i3 processors, DTLB0: 32-entries, 4-way set associative. ▪ Intel Core 2 Duo processors: DTLB0, 16 entries, DTLB1, 32 entries, 4 ways. ▪ Intel Atom processors: 8 entries, 4-way set associative. ▪ Pentium 4 and Intel Xeon processors: 64 entries, fully set associative; shared with small page data TLBs. ▪ Intel Core Duo, Intel Core Solo, Pentium M processor: 8 entries, fully associative. ▪ P6 family processors: 8 entries, 4-way set associative. ▪ Pentium processor: 8 entries, 4-way set associative; uses same TLB as used for 4-KByte pages in Pentium processors with MMX technology.
Second-level Unified TLB (4-KByte Pages)	▪ Intel Core i7, i5, i3 processor, STLB: 512-entries, 4-way set associative.

Cache or Buffer	Characteristics
Store Buffer	▪ Intel Core i7, i5, i3 processors: 32entries. ▪ Intel Core 2 Duo processors: 20 entries. ▪ Intel Atom processors: 8 entries, used for both WC and store buffers. ▪ Pentium 4 and Intel Xeon processors: 24 entries. ▪ Pentium M processor: 16 entries. ▪ P6 family processors: 12 entries. ▪ Pentium processor: 2 buffers, 1 entry each (Pentium processors with MMX technology have 4 buffers for 4 entries).
Write Combining (WC) Buffer	▪ Intel Core 2 Duo processors: 8 entries. ▪ Intel Atom processors: 8 entries, used for both WC and store buffers. ▪ Pentium 4 and Intel Xeon processors: 6 or 8 entries. ▪ Intel Core Duo, Intel Core Solo, Pentium M processors: 6 entries. ▪ P6 family processors: 4 entries.

NOTES:

1 Introduced to the IA-32 architecture in the Pentium 4 and Intel Xeon processors.

Intel 64 and IA-32 processors may implement four types of caches: the trace cache, the level 1 (L1) cache, the level 2 (L2) cache, and the level 3 (L3) cache. See Figure 11-1. Cache availability is described below:

- **Intel Core i7, i5, i3 processor Family and Intel Xeon processor Family based on Intel microarchitecture (Nehalem and Westmere)** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache. Each processor core has its own L1 and L2. The L3 cache is an inclusive, unified data and instruction cache, shared by all processor cores inside a physical package. No trace cache is implemented.
- **Intel Core 2 processor and Intel Xeon processor Family based on Intel Core microarchitecture** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache located on the processor chip; it is shared between two processor cores in a dual-core processor implementation. Quad-core processors have two L2, each shared by two processor cores. No trace cache is implemented.
- **Intel Atom processor** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache is located on the processor chip. No trace cache is implemented.
- **Intel Core Solo and Intel Core Duo processors** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache located on the processor chip. It is shared between two processor cores in a dual-core processor implementation. No trace cache is implemented.
- **Pentium 4 and Intel Xeon processors Based on Intel NetBurst microarchitecture** — The trace cache caches decoded instructions (μops) from the instruction decoder and the L1 cache contains data. The L2 and L3 caches are unified data and instruction caches located on the processor chip. Dualcore processors have two L2, one in each processor core. Note that the L3 cache is only implemented on some Intel Xeon processors.
- **P6 family processors** — The L1 cache is divided into two sections: one dedicated to caching instructions (pre-decoded instructions) and the other to caching data. The

L2 cache is a unified data and instruction cache located on the processor chip. P6 family processors do not implement a trace cache.

- **Pentium processors** — The L1 cache has the same structure as on P6 family processors. There is no trace cache. The L2 cache is a unified data and instruction cache external to the processor chip on earlier Pentium processors and implemented on the processor chip in later Pentium processors. For Pentium processors where the L2 cache is external to the processor, access to the cache is through the system bus.

For Intel Core i7 processors and processors based on Intel Core, Intel Atom, and Intel NetBurst microarchitectures, Intel Core Duo, Intel Core Solo and Pentium M processors, the cache lines for the L1 and L2 caches (and L3 caches if supported) are 64 bytes wide. The processor always reads a cache line from system memory beginning on a 64-byte boundary. (A 64-byte aligned cache line begins at an address with its 6 least-significant bits clear.) A cache line can be filled from memory with a 8-transfer burst transaction. The caches do not support partially-filled cache lines, so caching even a single double-word requires caching an entire line.

The L1 and L2 cache lines in the P6 family and Pentium processors are 32 bytes wide, with cache line reads from system memory beginning on a 32-byte boundary (5 least-significant bits of a memory address clear.) A cache line can be filled from memory with a 4-transfer burst transaction. Partially-filled cache lines are not supported.

The trace cache in processors based on Intel NetBurst microarchitecture is available in all execution modes: protected mode, system management mode (SMM), and real-address mode. The L1,L2, and L3 caches are also available in all execution modes; however, use of them must be handled carefully in SMM (see Section 26.4.2, “SMRAM Caching”).

The TLBs store the most recently used page-directory and page-table entries. They speed up memory accesses when paging is enabled by reducing the number of memory accesses that are required to read the page tables stored in system memory. The TLBs are divided into four groups: instruction TLBs for 4-KByte pages, data TLBs for 4-KByte pages; instruction TLBs for large pages (2-MByte, 4-MByte or 1-GByte pages), and data TLBs for large pages. The TLBs are normally active only in protected mode with paging enabled. When paging is disabled or the processor is in real-address mode, the TLBs maintain their contents until explicitly or implicitly flushed (see Section 11.9, “Invalidating the Translation Lookaside Buffers (TLBs)”).

...

11.9 INVALIDATING THE TRANSLATION LOOKASIDE BUFFERS (TLBS)

The processor updates its address translation caches (TLBs) transparently to software. Several mechanisms are available, however, that allow software and hardware to invalidate the TLBs either explicitly or as a side effect of another operation. Most details are given in Section 4.10.4, “Invalidation of TLBs and Paging-Structure Caches.” In addition, the following operations invalidate all TLB entries, irrespective of the setting of the G flag:

- Asserting or de-asserting the FLUSH# pin.
- (Pentium 4, Intel Xeon, and later processors only.) Writing to an MTRR (with a WRMSR instruction).
- Writing to control register CR0 to modify the PG or PE flag.

- (Pentium 4, Intel Xeon, and later processors only.) Writing to control register CR4 to modify the PSE, PGE, or PAE flag.
- Writing to control register CR4 to change the PCIDE flag from 1 to 0.

See Section 4.10, “Caching Translation Information,” for additional information about the TLBs.

...

11.11.2.3 Variable Range MTRRs

The Pentium 4, Intel Xeon, and P6 family processors permit software to specify the memory type for m variable-size address ranges, using a pair of MTRRs for each range. The number m of ranges supported is given in bits 7:0 of the IA32_MTRRCAP MSR (see Figure 11-5 in Section 11.11.1).

The first entry in each pair (IA32_MTRR_PHYSBASEn) defines the base address and memory type for the range; the second entry (IA32_MTRR_PHYSMASKn) contains a mask used to determine the address range. The “n” suffix is in the range 0 through m–1 and identifies a specific register pair.

...

11.11.9 Large Page Size Considerations

The MTRRs provide memory typing for a limited number of regions that have a 4 KByte granularity (the same granularity as 4-KByte pages). The memory type for a given page is cached in the processor’s TLBs. When using large pages (2 MBytes, 4 MBytes, or 1 GBytes), a single page-table entry covers multiple 4-KByte granules, each with a single memory type. Because the memory type for a large page is cached in the TLB, the processor can behave in an undefined manner if a large page is mapped to a region of memory that MTRRs have mapped with multiple memory types.

Undefined behavior can be avoided by insuring that all MTRR memory-type ranges within a large page are of the same type. If a large page maps to a region of memory containing different MTRR-defined memory types, the PCD and PWT flags in the page-table entry should be set for the most conservative memory type for that range. For example, a large page used for memory mapped I/O and regular memory is mapped as UC memory. Alternatively, the operating system can map the region using multiple 4-KByte pages each with its own memory type.

The requirement that all 4-KByte ranges in a large page are of the same memory type implies that large pages with different memory types may suffer a performance penalty, since they must be marked with the lowest common denominator memory type. The same consideration apply to 1 GByte pages, each of which may consist of multiple 2-Mbyte ranges.

The Pentium 4, Intel Xeon, and P6 family processors provide special support for the physical memory range from 0 to 4 MBytes, which is potentially mapped by both the fixed and variable MTRRs. This support is invoked when a Pentium 4, Intel Xeon, or P6 family processor detects a large page overlapping the first 1 MByte of this memory range with a memory type that conflicts with the fixed MTRRs. Here, the processor maps the memory range as multiple 4-KByte pages within the TLB. This operation insures correct behavior at the cost of performance. To avoid this performance penalty, operating-system software should reserve the large page option for regions of memory at addresses greater than or equal to 4 MBytes.

...

15. Updates to Chapter 22, Volume 3B

Change bars show changes to Chapter 22 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

22.2.1.1 Linear Accesses That Cause APIC-Access VM Exits

Whether a linear access to the APIC-access page causes an APIC-access VM exit depends in part of the nature of the translation used by the linear address:

- If the linear access uses a translation with a 4-KByte page, it causes an APIC-access VM exit.
- If the linear access uses a translation with a large page (2-MByte, 4-MByte, or 1-GByte), the access may or may not cause an APIC-access VM exit. Section 22.5.1 describes the treatment of such accesses that do not cause an APIC-access VM exits.

...

22.2.2.1 Guest-Physical Accesses That Might Not Cause APIC-Access VM Exits

Whether a guest-physical access to the APIC-access page causes an APIC-access VM exit depends on the nature of the EPT translation used by the guest-physical address and on how software is managing information cached from the EPT paging structures. The following items detail cases in which a guest-physical access to the APIC-access page might not an APIC-access VM exit:

- If the access uses a guest-physical address whose translation to the APIC-access page uses an EPT PDPTE that maps a 1-GByte page (because bit 7 of the EPT PDPTE is 1).

...

22.5.1 Linear Accesses to the APIC-Access Page Using Large-Page Translations

As noted in Section 22.2.1, a linear access to the APIC-access page using translation with a large page (2-MByte, 4-MByte, or 1-GByte) may or may not cause an APIC-access VM exit. If it does not and the access is not a VTPR access (see Section 22.2.4), the access operates on memory on the APIC-access page. Section 22.5.3 describes the treatment if there is no APIC-access VM exit and the access is a VTPR access.

...

22.5.3 VTPR Accesses

As noted in Section 22.2.4, a memory access is a VTPR access if all of the following hold: (1) the "use TPR shadow" VM-execution control is 1; (2) the access is not for an instruction fetch; (3) the access is at most 32 bits in width; and (4) the access is to offset 80H on the APIC-access page.

The treatment of VTPR accesses depends on the nature of the access:

- A linear VTPR access using a translation with a 4-KByte page does not cause an APIC-access VM exit. Instead, it is converted so that, instead of accessing offset 80H on the APIC-access page, it accesses offset 80H on the virtual-APIC page. Further details are provided in Section 22.5.3.1 to Section 22.5.3.3.
- A linear VTPR access using a translation with a large page (2-MByte, 4-MByte, or 1-GByte) may be treated in either of two ways:
  - It may operate on memory on the APIC-access page. The details in Section 22.5.3.1 to Section 22.5.3.3 do not apply.
  - It may be converted so that, instead of accessing offset 80H on the APIC-access page, it accesses offset 80H on the virtual-APIC page. Further details are provided in Section 22.5.3.1 to Section 22.5.3.3.

...

16. Updates to Chapter 23, Volume 3B

Change bars show changes to Chapter 23 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

23.2.2 Checks on Host Control Registers and MSRs

The following checks are performed on fields in the host-state area that correspond to control registers and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 20.8).[1]
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 20.8).
- On processors that support Intel 64 architecture, the CR3 field must be such that bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width must be 0.[2,3]
- On processors that support Intel 64 architecture, the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.
- If the "load IA32_PERF_GLOBAL_CTRL" VM-exit control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 30-3).
- If the "load IA32_PAT" VM-exit control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically,

1. The bits corresponding to CR0.NW (bit 29) and CR0.CD (bit 30) are never checked because the values of these bits are not changed by VM exit; see Section 24.5.1.
2. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
3. Bit 63 of the CR3 field in the host-state area must be 0. This is true even though, If CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 is used to determine whether cached translation information is invalidated.

each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).

...

23.2.4 Checks Related to Address-Space Size

On processors that support Intel 64 architecture, the following checks related to address-space size are performed on VMX controls and fields in the host-state area:

- If the logical processor is outside IA-32e mode (if IA32_EFER.LMA = 0) at the time of VM entry, the following must hold:
  - The "IA-32e mode guest" VM-entry control is 0.
  - The "host address-space size" VM-exit control is 0.
- If the logical processor is in IA-32e mode (if IA32_EFER.LMA = 1) at the time of VM entry, the "host address-space size" VM-exit control must be 1.
- If the "host address-space size" VM-exit control is 0, the following must hold:
  - The "IA-32e mode guest" VM-entry control is 0.
  - Bit 17 of the CR4 field (corresponding to CR4.PCIDE) is 0.
  - Bits 63:32 in the RIP field is 0.

...

23.3.1.1 Checks on Guest Control Registers, Debug Registers, and MSRs

The following checks are performed on fields in the guest-state area corresponding to control registers, debug registers, and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 20.8). The following are exceptions:
  - Bit 0 (corresponding to CR0.PE) and bit 31 (PG) are not checked if the "unrestricted guest" VM-execution control is 1.[1]
  - Bit 29 (corresponding to CR0.NW) and bit 30 (CD) are never checked because the values of these bits are not changed by VM entry; see Section 23.3.2.1.
- If bit 31 in the CR0 field (corresponding to PG) is 1, bit 0 in that field (PE) must also be 1.[2]
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 20.8).
- If the "load debug controls" VM-entry control is 1, bits reserved in the IA32_DEBUGCTL MSR must be 0 in the field for that register. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally.

---

1. "Unrestricted guest" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "unrestricted guest" VM-execution control were 0. See Section 21.6.2.
2. If the capability MSR IA32_VMX_CR0_FIXED1 reports that CR0.PE must be 1 in VMX operation, bit 0 in the CR0 field must be 1 unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- The following checks are performed on processors that support Intel 64 architecture:
  - If the "IA-32e mode guest" VM-entry control is 1, bit 31 in the CR0 field (corresponding to CR0.PG) and bit 5 in the CR4 field (corresponding to CR4.PAE) must each be 1.[1]
  - If the "IA-32e mode guest" VM-entry control is 0, bit 17 in the CR4 field (corresponding to CR4.PCIDE) must each be 0.
  - The CR3 field must be such that bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width are 0.[2,3]
  - If the "load debug controls" VM-entry control is 1, bits 63:32 in the DR7 field must be 0. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally (if they supported Intel 64 architecture).
  - The IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.

…

23.3.2.5 Updating Non-Register State

Section 25.3 describe how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM entries invalidate cached mappings:

- If the "enable VPID" VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP).
- VM entries are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the "enable VPID" VM-execution control is 1.

…

1. If the capability MSR IA32_VMX_CR0_FIXED1 reports that CR0.PG must be 1 in VMX operation, bit 31 in the CR0 field must be 1 unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
2. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
3. Bit 63 of the CR3 field in the guest-state area must be 0. This is true even though, If CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 is used to determine whether cached translation information is invalidated.

17. Updates to Chapter 24, Volume 3B

Change bars show changes to Chapter 24 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

24.2.1 Basic VM-Exit Information

Section 21.9.1 defines the basic VM-exit information fields. The following items detail their use.

- **Exit reason.**
  - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM exit. Appendix I lists the numbers used and their meaning.
  - The remainder of the field (bits 31:16) is cleared to 0 (certain SMM VM exits may set some of these bits; see Section 26.15.2.3).[1]
- **Exit qualification.** This field is saved for VM exits due to the following causes: debug exceptions; page-fault exceptions; start-up IPIs (SIPIs); system-management interrupts (SMIs) that arrive immediately after the retirement of I/O instructions; task switches; INVEPT; INVLPG; INVVPID; LGDT; LIDT; LLDT; LTR; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; control-register accesses; MOV DR; I/O instructions; MWAIT; accesses to the APIC-access page (see Section 22.2); and EPT violations. For all other VM exits, this field is cleared. The following items provide details:

For a debug exception, the exit qualification contains information about the debug exception. The information has the format given in Table 24-1.
...

Table 24-9 Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT

Bit Position(s)	Content
...	
11	Operand size: 0: 16-bit 1: 32-bit Undefined for VM exits from 64-bit mode.
...	

...

1. Bit 13 of this field is set on certain VM-entry failures; see Section 23.7.

24.5.1 Loading Host Control Registers, Debug Registers, MSRs

VM exits load new values for controls registers, debug registers, and some MSRs:

- CR0, CR3, and CR4 are loaded from the CR0 field, the CR3 field, and the CR4 field, respectively, with the following exceptions:
  - The following bits are not modified:
    - For CR0, ET, CD, NW; bits 63:32 (on processors that support Intel 64 architecture), 28:19, 17, and 15:6; and any bits that are fixed in VMX operation (see Section 20.8).[1]
    - For CR3, bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width (they are cleared to 0).[2] (This item applies only to processors that support Intel 64 architecture.)
    - For CR4, any bits that are fixed in VMX operation (see Section 20.8).
  - CR4.PAE is set to 1 if the "host address-space size" VM-exit control is 1.
  - CR4.PCIDE is set to 0 if the "host address-space size" VM-exit control is 0.

…

24.5.5 Updating Non-Register State

VM exits affect the non-register state of a logical processor as follows:

- A logical processor is always in the active state after a VM exit.
- Event blocking is affected as follows:
  - There is no blocking by STI or by MOV SS after a VM exit.
  - VM exits caused directly by non-maskable interrupts (NMIs) cause blocking by NMI (see Table 21-3). Other VM exits do not affect blocking by NMI. (See Section 24.1 for the case in which an NMI causes a VM exit indirectly.)
- There are no pending debug exceptions after a VM exit.

Section 25.3 describes how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM exits invalidate cached mappings:

- If the "enable VPID" VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP).
- VM exits are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the "enable VPID" VM-execution control is 1.

…

1. Bits 28:19, 17, and 15:6 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. CR0.ET is always 1 and the other bits are always 0.
2. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

18. Updates to Chapter 25, Volume 3B

Change bars show changes to Chapter 25 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

25.1 VIRTUAL PROCESSOR IDENTIFIERS (VPIDS)

The original architecture for VMX operation required VMX transitions to flush the TLBs and paging-structure caches. This ensured that translations cached for the old linear-address space would not be used after the transition.

Virtual-processor identifiers (**VPIDs**) introduce to VMX operation a facility by which a logical processor may cache information for multiple linear-address spaces. When VPIDs are used, VMX transitions may retain cached information and the logical processor switches to a different linear-address space.

Section 4.10 details the mechanisms by which a logical processor manages information cached for multiple address spaces. A logical processor may tag some cached information with a 16-bit VPID. This section specifies how the current VPID is determined at any point in time:

- The current VPID is 0000H in the following situations:
  - Outside VMX operation. (This includes operation in system-management mode under the default treatment of SMIs and SMM with VMX operation; see Section 26.14.)
  - In VMX root operation.
  - In VMX non-root operation when the "enable VPID" VM-execution control is 0.
- If the logical processor is in VMX non-root operation and the "enable VPID" VM-execution control is 1, the current VPID is the value of the VPID VM-execution control field in the VMCS. (VM entry ensures that this value is never 0000H; see Section 23.2.1.1.)

VPIDs and PCIDs (see Section 4.10.1) can be used concurrently. When this is done, the processor associates cached information with both a VPID and a PCID. Such information is used only if the current VPID and PCID **both** match those associated with the cached information.

...

25.3 CACHING TRANSLATION INFORMATION

Processors supporting Intel® 64 and IA-32 architectures may accelerate the address-translation process by caching on the processor data from the structures in memory that control that process. Such caching is discussed in Section 4.10, "Caching Translation Information" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.* The current section describes how this caching interacts with the VMX architecture.

The VPID and EPT features of the architecture for VMX operation augment this caching architecture. EPT defines the guest-physical address space and defines translations to

that address space (from the linear-address space) and from that address space (to the physical-address space). Both features control the ways in which a logical processor may create and use information cached from the paging structures.

Section 25.3.1 describes the different kinds of information that may be cached. Section 25.3.2 specifies when such information may be cached and how it may be used. Section 25.3.3 details how software can invalidate cached information.

25.3.1 Information That May Be Cached

Section 4.10, "Caching Translation Information" in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* identifies two kinds of translation-related information that may be cached by a logical processor: **translations**, which are mappings from linear page numbers to physical page frames, and **paging-structure caches**, which map the upper bits of a linear page number to information from the paging-structure entries used to translate linear addresses matching those upper bits.

The same kinds of information may be cached when VPIDs and EPT are in use. A logical processor may cache and use such information based on its function. Information with different functionality is identified as follows:

- **Linear mappings**.[1] There are two kinds:
  - Linear translations. Each of these is a mapping from a linear page number to the physical page frame to which it translates, along with information about access privileges and memory typing.
  - Linear paging-structure-cache entries. Each of these is a mapping from the upper portion of a linear address to the physical address of the paging structure used to translate the corresponding region of the linear-address space, along with information about access privileges. For example, bits 47:39 of a linear address would map to the address of the relevant page-directory-pointer table.

  Linear mappings do not contain information from any EPT paging structure.
- **Guest-physical mappings**.[2] There are two kinds:
  - Guest-physical translations. Each of these is a mapping from a guest-physical page number to the physical page frame to which it translates, along with information about access privileges and memory typing.
  - Guest-physical paging-structure-cache entries. Each of these is a mapping from the upper portion of a guest-physical address to the physical address of the EPT paging structure used to translate the corresponding region of the guest-physical address space, along with information about access privileges.

  The information in guest-physical mappings about access privileges and memory typing is derived from EPT paging structures.
- **Combined mappings**.[3] There are two kinds:
  - Combined translations. Each of these is a mapping from a linear page number to the physical page frame to which it translates, along with information about access privileges and memory typing.

1. Earlier versions of this manual used the term "VPID-tagged" to identify linear mappings.
2. Earlier versions of this manual used the term "EPTP-tagged" to identify guest-physical mappings.
3. Earlier versions of this manual used the term "dual-tagged" to identify combined mappings.

— Combined paging-structure-cache entries. Each of these is a mapping from the upper portion of a linear address to the physical address of the paging structure used to translate the corresponding region of the linear-address space, along with information about access privileges.

The information in combined mappings about access privileges and memory typing is derived from both guest paging structures and EPT paging structures.

25.3.2 Creating and Using Cached Translation Information

The following items detail the creation of the mappings described in the previous section:[1]

- The following items describe the creation of mappings while EPT is not in use (including execution outside VMX non-root operation):
  - Linear mappings may be created. They are derived from the paging structures referenced (directly or indirectly) by the current value of CR3 and are associated with the current VPID and the current PCID.
  - No linear mappings are created with information derived from paging-structure entries that are not present (bit 0 is 0) or that set reserved bits. For example, if a PTE is not present, no linear mapping are created for any linear page number whose translation would use that PTE.
  - No guest-physical or combined mappings are created while EPT is not in use.
- The following items describe the creation of mappings while EPT is in use:
  - Guest-physical mappings may be created. They are derived from the EPT paging structures referenced (directly or indirectly) by bits 51:12 of the current EPTP. These 40 bits contain the address of the EPT-PML4-table. (the notation **EP4TA** refers to those 40 bits). Newly created guest-physical mappings are associated with the current EP4TA.
  - Combined mappings may be created. They are derived from the EPT paging structures referenced (directly or indirectly) by the current EP4TA. If CR0.PG = 1, they are also derived from the paging structures referenced (directly or indirectly) by the current value of CR3. They are associated with the current VPID, the current PCID, and the current EP4TA.[2] No combined paging-structure-cache entries are created if CR0.PG = 0.[3]
  - No guest-physical mappings or combined mappings are created with information derived from EPT paging-structure entries that are not present (bits 2:0 are all 0) or that are misconfigured (see Section 25.2.3.1).
  - No combined mappings are created with information derived from guest paging-structure entries that are not present or that set reserved bits.

---

1. This section associated cached information with the current VPID and PCID. If PCIDs are not supported or are not being used (e.g., because CR4.PCIDE = 0), all the information is implicitly associated with PCID 000H; see Section 4.10.1, "Process-Context Identifiers (PCIDs)," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.
2. At any given time, a logical processor may be caching combined mappings for a VPID and a PCID that are associated with different EP4TAs. Similarly, it may be caching combined mappings for an EP4TA that are associated with different VPIDs and PCIDs.
3. If the capability MSR IA32_VMX_CR0_FIXED1 reports that CR0.PG must be 1 in VMX operation, CR0.PG can be 0 in VMX non-root operation only if the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

— No linear mappings are created while EPT is in use.

The following items detail the use of the various mappings:

- If EPT is not in use (e.g., when outside VMX non-root operation), a logical processor may use cached mappings as follows:
  - For accesses using linear addresses, it may use linear mappings associated with the current VPID and the current PCID.
  - No guest-physical or combined mappings are used while EPT is not in use.
- If EPT is in use, a logical processor may use cached mappings as follows:
  - For accesses using linear addresses, it may use combined mappings associated with the current VPID, the current PCID, and the current EP4TA.
  - For accesses using guest-physical addresses, it may use guest-physical mappings associated with the current EP4TA.
  - No linear mappings are used while EPT is in use.

25.3.3 Invalidating Cached Translation Information

Software modifications of paging structures (including EPT paging structures) may result in inconsistencies between those structures and the mappings cached by a logical processor. Certain operations invalidate information cached by a logical processor and can be used to eliminate such inconsistencies.

25.3.3.1 Operations that Invalidate Cached Mappings

The following operations invalidate cached mappings as indicated:

- Operations that architecturally invalidate entries in the TLBs or paging-structure caches independent of VMX operation (e.g., the INVLPG instruction) invalidate linear mappings and combined mappings.[1] They are required to do so only for the current VPID (but, for combined mappings, all EP4TAs). Linear mappings for the current VPID are invalidated even if EPT is in use.[2] Combined mappings for the current VPID are invalidated even if EPT is not in use.[3]
- An EPT violation invalidates any guest-physical mappings (associated with the current EP4TA) that would be used to translate the guest-physical address that caused the EPT violation. If that guest-physical address was the translation of a linear address, the EPT violation also invalidates any combined mappings for that linear address associated with the current PCID, the current VPID and the current EP4TA.

---

1. See Section 4.10.4, "Invalidation of TLBs and Paging-Structure Caches," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* for an enumeration of operations that architecturally invalidate entries in the TLBs and paging-structure caches independent of VMX operation.
2. While no linear mappings are created while EPT is in use, a logical processor may retain, while EPT is in use, linear mappings (for the same VPID as the current one) there were created earlier, when EPT was not in use.
3. While no combined mappings are created while EPT is not in use, a logical processor may retain, while EPT is in not use, combined mappings (for the same VPID as the current one) there were created earlier, when EPT was in use.

- If the “enable VPID” VM-execution control is 0, VM entries and VM exits invalidate linear mappings and combined mappings associated with VPID 0000H (for all PCIDs). Combined mappings for VPID 0000H are invalidated for all EP4TAs.
- Execution of the INVVPID instruction invalidates linear mappings and combined mappings. Invalidation is based on instruction operands, called the INVVPID type and the INVVPID descriptor. Four INVVPID types are currently defined:
  - **Individual-address.** If the INVVPID type is 0, the logical processor invalidates linear mappings and combined mappings associated with the VPID specified in the INVVPID descriptor and that would be used to translate the linear address specified in of the INVVPID descriptor. Linear mappings and combined mappings for that VPID and linear address are invalidated for all PCIDs and, for combined mappings, all EP4TAs. (The instruction may also invalidate mappings associated with other VPIDs and for other linear addresses.)
  - **Single-context.** If the INVVPID type is 1, the logical processor invalidates all linear mappings and combined mappings associated with the VPID specified in the INVVPID descriptor. Linear mappings and combined mappings for that VPID are invalidated for all PCIDs and, for combined mappings, all EP4TAs. (The instruction may also invalidate mappings associated with other VPIDs.)
  - **All-context.** If the INVVPID type is 2, the logical processor invalidates linear mappings and combined mappings associated with all VPIDs except VPID 0000H and with all PCIDs. (In some cases, it may invalidate linear mappings with VPID 0000H as well.) Combined mappings are invalidated for all EP4TAs.
  - **Single-context-retaining-globals.** If the INVVPID type is 3, the logical processor invalidates linear mappings and combined mappings associated with the VPID specified in the INVVPID descriptor. Linear mappings and combined mappings for that VPID are invalidated for all PCIDs and, for combined mappings, all EP4TAs. The logical processor is **not** required to invalidate information that was used for **global** translations (although it may do so). See Section 4.10, “Caching Translation Information” for details regarding global translations. (The instruction may invalidate mappings associated with other VPIDs.)

  See Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B* for details of the INVVPID instruction. See Section 25.3.3.3 for guidelines regarding use of this instruction.
- Execution of the INVEPT instruction invalidates guest-physical mappings and combined mappings. Invalidation is based on instruction operands, called the INVEPT type and the INVEPT descriptor. Two INVEPT types are currently defined:
  - **Single-context.** If the INVEPT type is 1, the logical processor invalidates all guest-physical mappings and combined mappings associated with the EP4TA specified in the INVEPT descriptor. Combined mappings for that EP4TA are invalidated for all VPIDs and all PCIDs. (The instruction may invalidate mappings associated with other EP4TAs.)
  - **All-context.** If the INVEPT type is 2, the logical processor invalidates guest-physical mappings and combined mappings associated with all EP4TAs (and, for combined mappings, for all VPIDs and PCIDs).

  See Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B* for details of the INVEPT instruction. See Section 25.3.3.4 for guidelines regarding use of this instruction.
- A power-up or a reset invalidates all linear mappings, guest-physical mappings, and combined mappings.

25.3.3.2 Operations that Need Not Invalidate Cached Mappings

The following items detail cases of operations that are not required to invalidate certain cached mappings:

- Operations that architecturally invalidate entries in the TLBs or paging-structure caches independent of VMX operation are not required to invalidate any guest-physical mappings.
- The INVVPID instruction is not required to invalidate any guest-physical mappings.
- The INVEPT instruction is not required to invalidate any linear mappings.
- VMX transitions are not required to invalidate any guest-physical mappings. If the "enable VPID" VM-execution control is 1, VMX transitions are not required to invalidate any linear mappings or combined mappings.
- The VMXOFF and VMXON instructions are not required to invalidate any linear mappings, guest-physical mappings, or combined mappings.

A logical processor may invalidate any cached mappings at any time. For this reason, the operations identified above may invalidate the indicated mappings despite the fact that doing so is not required.

25.3.3.3 Guidelines for Use of the INVVPID Instruction

The need for VMM software to use the INVVPID instruction depends on how that software is virtualizing memory (e.g., see Section 28.3, "Memory Virtualization").

If EPT is not in use, it is likely that the VMM is virtualizing the guest paging structures. Such a VMM may configure the VMCS so that all or some of the operations that invalidate entries the TLBs and the paging-structure caches (e.g., the INVLPG instruction) cause VM exits. If VMM software is emulating these operations, it may be necessary to use the INVVPID instruction to ensure that the logical processor's TLBs and the paging-structure caches are appropriately invalidated.

Requirements of when software should use the INVVPID instruction depend on the specific algorithm being used for page-table virtualization. The following items provide guidelines for software developers:

- Emulation of the INVLPG instruction may require execution of the INVVPID instruction as follows:
  - The INVVPID type is individual-address (0).
  - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.
  - The linear address in the INVVPID descriptor is that of the operand of the INVLPG instruction being emulated.
- Some instructions invalidate all entries in the TLBs and paging-structure caches—except for global translations. An example is the MOV to CR3 instruction. (See Section 4.10, "Caching Translation Information" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* for details regarding global translations.) Emulation of such an instruction may require execution of the INVVPID instruction as follows:
  - The INVVPID type is single-context-retaining-globals (3).
  - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.

- Some instructions invalidate all entries in the TLBs and paging-structure caches—including for global translations. An example is the MOV to CR4 instruction if the value of value of bit 4 (page global enable—PGE) is changing. Emulation of such an instruction may require execution of the INVVPID instruction as follows:
  - The INVVPID type is single-context (1).
  - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.

If EPT is not in use, the logical processor associates all mappings it creates with the current VPID, and it will use such mappings to translate linear addresses. For that reason, a VMM should not use the same VPID for different non-EPT guests that use different page tables. Doing so may result in one guest using translations that pertain to the other.

If EPT is in use, the instructions enumerated above might not be configured to cause VM exits and the VMM might not be emulating them. In that case, executions of the instructions by guest software properly invalidate the required entries in the TLBs and paging-structure caches (see Section 25.3.3.1); execution of the INVVPID instruction is not required.

If EPT is in use, the logical processor associates all mappings it creates with the value of bits 51:12 of current EPTP. If a VMM uses different EPTP values for different guests, it may use the same VPID for those guests. Doing so cannot result in one guest using translations that pertain to the other.

The following guidelines apply more generally and are appropriate even if EPT is in use:

- As detailed in Section 22.2.1.1, an access to the APIC-access page might not cause an APIC-access VM exit if software does not properly invalidate information that may be cached from the paging structures. If, at one time, the current VPID on a logical processor was a non-zero value X, it is recommended that software use the INVVPID instruction with the “single-context” INVVPID type and with VPID X in the INVVPID descriptor before a VM entry on the same logical processor that establishes VPID X and either (a) the “virtualize APIC accesses” VM-execution control was changed from 0 to 1; or (b) the value of the APIC-access address was changed.
- Software can use the INVVPID instruction with the “all-context” INVVPID type immediately after execution of the VMXON instruction or immediately prior to execution of the VMXOFF instruction. Either prevents potentially undesired retention of information cached from paging structures between separate uses of VMX operation.

25.3.3.4 Guidelines for Use of the INVEPT Instruction

The following items provide guidelines for use of the INVEPT instruction to invalidate information cached from the EPT paging structures.

- Software should use the INVEPT instruction with the “single-context” INVEPT type after making any of the following changes to an EPT paging-structure entry (the INVEPT descriptor should contain an EPTP value that references — directly or indirectly — the modified EPT paging structure):
  - Changing any of the privilege bits 2:0 from 1 to 0.
  - Changing the physical address in bits 51:12.
  - For an EPT PDPTE or an EPT PDE, changing bit 7 (which determines whether the entry maps a page).

  — For the **last** EPT paging-structure entry used to translate a guest-physical address (either an EPT PDE with bit 7 set to 1 or an EPT PTE), changing either bits 5:3 or bit 6. (These bits determine the effective memory type of accesses using that EPT paging-structure entry; see Section 25.2.4.)

- Software may use the INVEPT instruction after modifying a present EPT paging-structure entry to change any of the privilege bits 2:0 from 0 to 1. Failure to do so may cause an EPT violation that would not otherwise occur. Because an EPT violation invalidates any mappings that would be used by the access that caused the EPT violation (see Section 25.3.3.1), an EPT violation will not recur if the original access is performed again, even if the INVEPT instruction is not executed.
- Because a logical processor does not cache any information derived from EPT paging-structure entries that are not present or misconfigured (see Section 25.2.3.1), it is not necessary to execute INVEPT following modification of an EPT paging-structure entry that had been not present or misconfigured.
- As detailed in Section 22.2.1.1 and Section 22.2.2.1, an access to the APIC-access page might not cause an APIC-access VM exit if software does not properly invalidate information that may be cached from the EPT paging structures. If EPT was in use on a logical processor at one time with EPTP X, it is recommended that software use the INVEPT instruction with the “single-context” INVEPT type and with EPTP X in the INVEPT descriptor before a VM entry on the same logical processor that enables EPT with EPTP X and either (a) the “virtualize APIC accesses” VM-execution control was changed from 0 to 1; or (b) the value of the APIC-access address was changed.
- Software can use the INVEPT instruction with the “all-context” INVEPT type immediately after execution of the VMXON instruction or immediately prior to execution of the VMXOFF instruction. Either prevents potentially undesired retention of information cached from EPT paging structures between separate uses of VMX operation.

In a system containing more than one logical processor, software must account for the fact that information from an EPT paging-structure entry may be cached on logical processors other than the one that modifies that entry. The process of propagating the changes to a paging-structure entry is commonly referred to as “TLB shootdown.” A discussion of TLB shootdown appears in Section 4.10.5, “Propagation of Paging-Structure Changes to Multiple Processors,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

19. Updates to Chapter 26, Volume 3B

Change bars show changes to Chapter 26 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

26.3.2 Exiting From SMM

The only way to exit SMM is to execute the RSM instruction. The RSM instruction is only available to the SMI handler; if the processor is not in SMM, attempts to execute the RSM instruction result in an invalid-opcode exception (#UD) being generated.

The RSM instruction restores the processor's context by loading the state save image from SMRAM back into the processor's registers. The processor then returns an SMIACK transaction on the system bus and returns program control back to the interrupted program.

Upon successful completion of the RSM instruction, the processor signals external hardware that SMM has been exited. For the P6 family processors, an SMI acknowledge transaction is generated on the system bus and the multiplexed status signal EXF4 is no longer generated on bus cycles. For the Pentium and Intel486 processors, the SMIACT# pin is deserted.

If the processor detects invalid state information saved in the SMRAM, it enters the shutdown state and generates a special bus cycle to indicate it has entered shutdown state. Shutdown happens only in the following situations:

- A reserved bit in control register CR4 is set to 1 on a write to CR4. This error should not happen unless SMI handler code modifies reserved areas of the SMRAM saved state map (see Section 26.4.1). CR4 is saved in the state map in a reserved location and cannot be read or modified in its saved state.
- An illegal combination of bits is written to control register CR0, in particular PG set to 1 and PE set to 0, or NW set to 1 and CD set to 0.
- CR4.PCIDE would be set to 1 and IA32_EFER.LMA to 0.
- (For the Pentium and Intel486 processors only.) If the address stored in the SMBASE register when an RSM instruction is executed is not aligned on a 32-KByte boundary. This restriction does not apply to the P6 family processors.

...

26.4.1 SMRAM State Save Map

When an IA-32 processor that does not support Intel 64 architecture initially enters SMM, it writes its state to the state save area of the SMRAM. The state save area begins at [SMBASE + 8000H + 7FFFH] and extends down to [SMBASE + 8000H + 7E00H]. Table 26-1 shows the state save map. The offset in column 1 is relative to the SMBASE value plus 8000H. Reserved spaces should not be used by software.

...

The following registers are saved (but not readable) and restored upon exiting SMM:

- Control register CR4. (This register is cleared to all 0s when entering SMM).

- The hidden segment descriptor information stored in segment registers CS, DS, ES, FS, GS, and SS.

…

26.5 SMI HANDLER EXECUTION ENVIRONMENT

After saving the current context of the processor, the processor initializes its core registers to the values shown in Table 26-4. Upon entering SMM, the PE and PG flags in control register CR0 are cleared, which places the processor is in an environment similar to real-address mode. The differences between the SMM execution environment and the real-address mode execution environment are as follows:

- The addressable SMRAM address space ranges from 0 to FFFFFFFFH (4 GBytes). (The physical address extension — enabled with the PAE flag in control register CR4 — is not supported in SMM.)

…

26.15.2.5 Updating Non-Register State

SMM VM exits affect the non-register state of a logical processor as follows:

- SMM VM exits cause non-maskable interrupts (NMIs) to be blocked; they may be unblocked through execution of IRET or through a VM entry (depending on the value loaded for the interruptibility state and the setting of the “virtual NMIs” VM-execution control).
- SMM VM exits cause SMIs to be blocked; they may be unblocked by a VM entry that returns from SMM (see Section 26.15.4).

SMM VM exits invalidate linear mappings and combined mappings associated with VPID 0000H for all PCIDs. Combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP; see Section 25.3). (Ordinary VM exits are not required to perform such invalidation if the “enable VPID” VM-execution control is 1; see Section 24.5.5.)

…

26.15.4.5 Loading Guest State

VM entries that return from SMM load the SMBASE register from the SMBASE field.

VM entries that return from SMM invalidate linear mappings and combined mappings associated with all VPIDs. Combined mappings are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP; see Section 25.3). (Ordinary VM entries are required to perform such invalidation only for VPID 0000H and are not required to do even that if the “enable VPID” VM-execution control is 1; see Section 23.3.2.5.)

…

20. Updates to Chapter 27, Volume 3B

Change bars show changes to Chapter 27 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

27.4 USING VMX INSTRUCTIONS

VMX instructions are allowed only in VMX root operation. An attempt to execute a VMX instruction in VMX non-root operation causes a VM exit.

Processors perform various checks while executing any VMX instruction. They follow well-defined error handling on failures. VMX instruction execution failures detected before loading of a guest state are handled by the processor as follows:

- If the working-VMCS pointer is not valid, the instruction fails by setting RFLAGS.CF to 1.
- If the working-VMCS pointer is valid, RFLAGS.ZF is set to 1 and the proper error-code is saved in the VM-instruction error field of the working-VMCS.

Software is required to check RFLAGS.CF and RFLAGS.ZF to determine the success or failure of VMX instruction executions.

The following items provide details regarding use of the VM-entry instructions (VMLAUNCH and VMRESUME):

- If the working-VMCS pointer is valid, the state of the working VMCS may cause the VM-entry instruction to fail. RFLAGS.ZF is set to 1 and one of the following values is saved in the VM-instruction error field:
  - 4: VMLAUNCH with non-clear VMCS.
    If this error occurs, software can avoid the error by executing VMRESUME.
  - 5: VMRESUME with non-launched VMCS.
    If this error occurs, software can avoid the error by executing VMLAUNCH.
  - 6: VMRESUME after VMXOFF.[1]
    If this error occurs, software can avoid the error by executing the following sequence of instructions:

    ```
    VMPTRST ⟨working-VMCS pointer⟩
    VMCLEAR ⟨working-VMCS pointer⟩
    VMPTRLD ⟨working-VMCS pointer⟩
    VMLAUNCH
    ```

    (VMPTRST may not be necessary is software already knows the working-VMCS pointer.)
- If none of the above errors occur, the processor checks on the VMX controls and host-state area. If any of these checks fail, the VM-entry instruction fails. RFLAGS.ZF is set to 1 and either 7 (VM entry with invalid control field(s)) or 8 (VM entry with invalid host-state field(s)) is saved in the VM-instruction error field.
- After a VM-entry instruction (VMRESUME or VMLAUNCH) successfully completes the general checks and checks on VMX controls and the host-state area (see Section

1. Earlier versions of this manual described this error as "VMRESUME with a corrupted VMCS".

23.2), any errors encountered while loading of guest-state (due to bad guest-state or bad MSR loading) causes the processor to load state from the host-state area of the working VMCS as if a VM exit had occurred (see Section 27.7).

This failure behavior differs from that of VM exits in that no guest-state is saved to the guest-state area. A VMM can detect its VM-exit handler was invoked by such a failure by checking bit 31 (for 1) in the exit reason field of the working VMCS and further identify the failure by using the exit qualification field.

See Chapter 23 for more details about the VM-entry instructions.

...

21. Updates to Chapter 30, Volume 3B

Change bars show changes to Chapter 30 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

30.6.2 Performance Monitoring Facility in the Uncore

The "uncore" in Intel microarchitecture codename Nehalem refers to subsystems in the physical processor package that are shared by multiple processor cores. Some of the sub-systems in the uncore include the L3 cache, Intel QuickPath Interconnect link logic, and integrated memory controller. The performance monitoring facilities inside the uncore operates in the same clock domain as the uncore (U-clock domain), which is usually different from the processor core clock domain. The uncore performance monitoring facilities described in this section apply to Intel Xeon processor 5500 series and processors with the following CPUID signatures: 06_1AH, 06_1EH, 06_1FH (see Appendix B). An overview of the uncore performance monitoring facilities is described separately.

...

30.6.3 Intel Xeon Processor 7500 Series Performance Monitoring Facility

The performance monitoring facility in the processor core of Intel Xeon processor 7500 series are the same as those supported in Intel Xeon processor 5500 series. The uncore subsystem in Intel Xeon processor 7500 series are significantly different The uncore performance monitoring facility consist of many distributed units associated with individual logic control units (referred to as boxes) within the uncore subsystem. A high level block diagram of the various box units of the uncore is shown in Figure Figure 30-23.

Uncore PMUs are programmed via MSR interfaces. Each of the distributed uncore PMU units have several general-purpose counters. Each counter requires an associated event select MSR, and may require additional MSRs to configure sub-event conditions. The uncore PMU MSRs associated with each box can be categorized based on its functional scope: per-counter, per-box, or global across the uncore. The number counters available in each box type are different. Each box generally provides a set of MSRs to enable/disable, check status/overflow of multiple counters within each box.

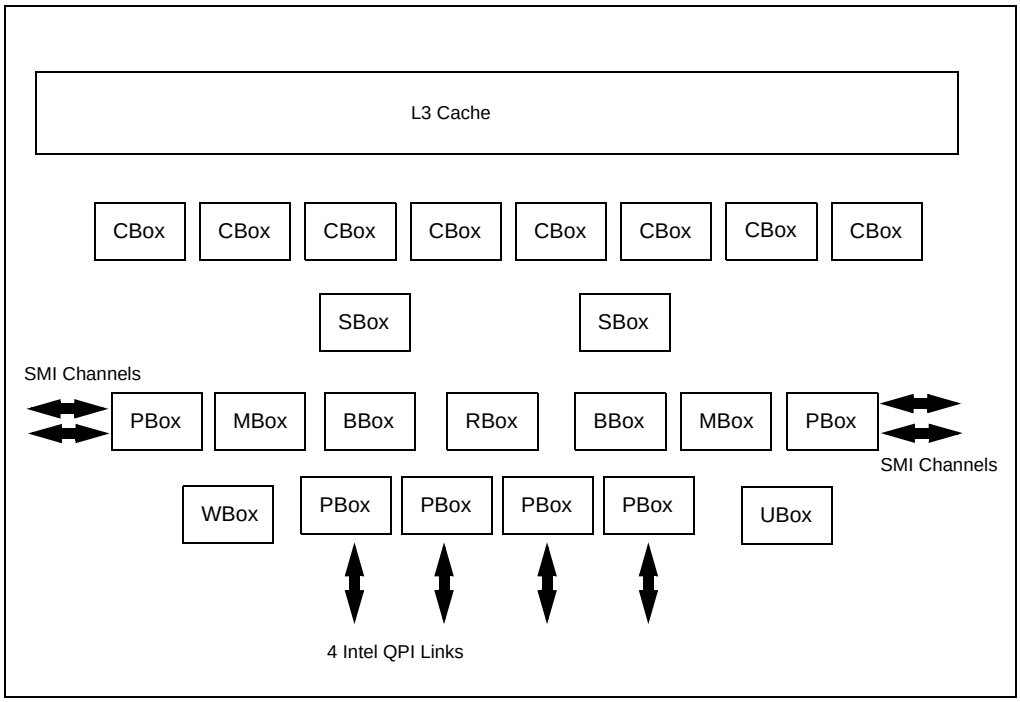

Figure 30-23 Distributed Units of the Uncore of Intel Xeon Processor 7500 Series

Table 30-17 summarizes the number MSRs for uncore PMU for each box.

Table 30-17 Uncore PMU MSR Summary

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	8	6	48	Yes	per-box	None
S-Box	2	4	48	Yes	per-box	Match/Mask
B-Box	2	4	48	Yes	per-box	Match/Mask
M-Box	2	6	48	Yes	per-box	Yes
R-Box	1	16 (2 port, 8 per port)	48	Yes	per-box	Yes
W-Box	1	4	48	Yes	per-box	None
		1	48	No	per-box	None
U-Box	1	1	48	Yes	uncore	None

The W-Box provides 4 general-purpose counters, each requiring an event select configuration MSR, similar to the general-purpose counters in other boxes. There is also a fixed-function counter that increments clockticks in the uncore clock domain.

For C,S,B,M,R, and W boxes, each box provides an MSR to enable/disable counting, configuring PMI of multiple counters within the same box, this is somewhat similar the “global control“ programming interface, IA32_PERF_GLOBAL_CTRL, offered in the core

PMU. Similarly status information and counter overflow control for multiple counters within the same box are also provided in C,S,B,M,R, and W boxes.

In the U-Box, MSR_U_PMON_GLOBAL_CTL provides overall uncore PMU enable/disable and PMI configuration control. The scope of status information in the U-box is at per-box granularity, in contrast to the per-box status information MSR (in the C,S,B,M,R, and W boxes) providing status information of individual counter overflow. The difference in scope also apply to the overflow control MSR in the U-Box versus those in the other Boxes.

The individual MSRs that provide uncore PMU interfaces are listed in Appendix B. Table B-7 under the general naming style of MSR_%box#%_PMON_%scope_function%, where %box#% designates the type of box and zero-based index if there are more the one box of the same type, %scope_function% follows the examples below:

- Multi-counter enabling MSRs: MSR_U_PMON_GLOBAL_CTL, MSR_S0_PMON_BOX_CTL, MSR_C7_PMON_BOX_CTL, etc.
- Multi-counter status MSRs: MSR_U_PMON_GLOBAL_STATUS, MSR_S0_PMON_BOX_STATUS, MSR_C7_PMON_BOX_STATUS, etc.
- Multi-counter overflow control MSRs: MSR_U_PMON_GLOBAL_OVF_CTL, MSR_S0_PMON_BOX_OVF_CTL, MSR_C7_PMON_BOX_OVF_CTL, etc.
- Performance counters MSRs: the scope is implicitly per counter, e.g. MSR_U_PMON_CTR, MSR_S0_PMON_CTR0, MSR_C7_PMON_CTR5, etc
- Event select MSRs: the scope is implicitly per counter, e.g. MSR_U_PMON_EVNT_SEL, MSR_S0_PMON_EVNT_SEL0, MSR_C7_PMON_EVNT_SEL5, etc
- Sub-control MSRs: the scope is implicitly per-box granularity, e.g. MSR_M0_PMON_TIMESTAMP, MSR_R0_PMON_IPERF0_P1, MSR_S1_PMON_MATCH.

Details of uncore PMU MSR bit field definitions can be found in a separate document "Intel Xeon Processor 7500 Series Uncore Performance Monitoring Guide".

...

22. Updates to Appendix A, Volume 3B

Change bars show changes to Appendix A of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

…

Table A-2 Non-Architectural Performance Events In the Processor Core for Intel Core i7 Processor and Intel Xeon Processor 5500 Series

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
…				
BBH	01H	OFF_CORE_RESPONSE_1	See Section 30.7, "Performance Monitoring for Processors based on Intel® microarchitecture CodeName Westmere".	Requires programming MSR 01A7H
…				

…

23. Updates to Appendix B, Volume 3B

Change bars show changes to Appendix B of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

…

Table B-1 CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
…	
06_2EH	Intel Xeon Processor 7500 series
06_25H, 06_2CH	Intel Xeon Processor 5600 series, Intel Core i7, i5 and i3 Processor
…	

…

Table B-2 Architectural MSRs

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
...				
1BH	27	IA32_APIC_BASE (APIC_BASE)		06_01H
		7:0	Reserved	
		8	BSP flag (R/W)	
		9	Reserved	
		10	Enable x2APIC mode	06_1AH
		11	APIC Global Enable (R/W)	
		(MAXPHYWID - 1):12	APIC Base (R/W)	
		63: MAXPHYWID	Reserved	
...				
1A0H	416	IA32_MISC_ENABLE	**Enable Misc. Processor Features. (R/W)** Allows a variety of processor functions to be enabled and disabled.	
		0	**Fast-Strings Enable.** When set, the fast-strings feature (for REP MOVS and REP STORS) is enabled (default); when clear, fast-strings are disabled.	0F_0H
		2:1	Reserved.	
		3	**Automatic Thermal Control Circuit Enable. (R/W)** 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows the processor to automatically reduce power consumption in response to TCC activation.. 0 = Disabled (default).	0F_0H
		6:4	Reserved	

		7	**Performance Monitoring Available. (R)** 1 = Performance monitoring enabled 0 = Performance monitoring disabled	0F_0H
		10:8	Reserved	
		11	**Branch Trace Storage Unavailable. (RO)** 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported	0F_0H
		12	**Precise Event Based Sampling (PEBS) Unavailable. (RO)** 1 = PEBS is not supported; 0 = PEBS is supported.	06_0FH
		15:13	Reserved	
		16	**Enhanced Intel SpeedStep Technology Enable. (R/W)** 0= Enhanced Intel SpeedStep Technology disabled 1 = Enhanced Intel SpeedStep Technology enabled	06_0DH
		17	Reserved	

		18	**ENABLE MONITOR FSM.** (R/W) When this bit is set to 0, the MONITOR feature flag is not set (CPUID.01H:ECX[bit 3] = 0). This indicates that MONITOR/MWAIT are not supported. Software attempts to execute MONITOR/MWAIT will cause #UD when this bit is 0. When this bit is set to 1 (default), MONITOR/MWAIT are supported (CPUID.01H:ECX[bit 3] = 1). If the SSE3 feature flag ECX[0] is not set (CPUID.01H:ECX[bit 0] = 0), the OS must not attempt to alter this bit. BIOS must leave it in the default state. Writing this bit when the SSE3 feature flag is set to 0 may generate a #GP exception.	0F_03H
		21:19	Reserved	

		22	**Limit CPUID Maxval. (R/W)** When this bit is set to 1, CPUID.00H returns a maximum value in EAX[7:0] of 3. BIOS should contain a setup question that allows users to specify when the installed OS does not support CPUID functions greater than 3. Before setting this bit, BIOS must execute the CPUID.0H and examine the maximum value returned in EAX[7:0]. If the maximum value is greater than 3, the bit is supported. Otherwise, the bit is not supported. Writing to this bit when the maximum value is greater than 3 may generate a #GP exception. Setting this bit may cause unexpected behavior in software that depends on the availability of CPUID leaves greater than 3.	0F_03H
		23	**xTPR Message Disable. (R/W)** When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority.	if CPUID.01H:ECX[14] = 1
		33:24	Reserved	

		34	**XD Bit Disable. (R/W)** When set to 1, the Execute Disable Bit feature (XD Bit) is disabled and the XD Bit extended feature flag will be clear (CPUID.80000001H: EDX[20]=0). When set to a 0 (default), the Execute Disable Bit feature (if available) allows the OS to enable PAE paging and take advantage of data only pages. BIOS must not alter the contents of this bit location, if XD bit is not supported.. Writing this bit to 1 when the XD Bit extended feature flag is set to 0 may generate a #GP exception.	if CPUID.80000001H:EDX[20] = 1
		63:35	Reserved	
...				
1F2H	498	IA32_SMRR_PHYSBASE	**SMRR Base Address. (Writeable only in SMM)** Base address of SMM memory range.	06_1AH
		7:0	Type. Specifies memory type of the range.	
		11:8	Reserved.	
		31:12	**PhysBase.** SMRR physical Base Address.	
		63:32	Reserved.	
1F3H	499	IA32_SMRR_PHYSMASK	**SMRR Range Mask. (Writeable only in SMM)** Range Mask of SMM memory range.	06_1AH
		10:0	Reserved.	
		11	**Valid.** Enable range mask	
		31:12	**PhysMask.** SMRR address range mask.	
		63:32	Reserved.	
1F8H	504	IA32_PLATFORM_DCA_CAP	DCA Capability (R)	06_0FH

1F9H	505	IA32_CPU_DCA_CAP	If set, CPU supports Prefetch-Hint type.	
1FAH	506	IA32_DCA_0_CAP	DCA type 0 Status and Control register	06_2EH
		0	DCA_ACTIVE: Set by HW when DCA is fuse-enabled and no defeatures are set.	06_2EH
		2:1	TRANSACTION	06_2EH
		6:3	DCA_TYPE	06_2EH
		10:7	DCA_QUEUE_SIZE	06_2EH
		12:11	Reserved.	06_2EH
		16:13	DCA_DELAY: Writes will update the register but have no HW side-effect.	06_2EH
		23:17	Reserved.	06_2EH
		24	SW_BLOCK: SW can request DCA block by setting this bit.	06_2EH
		25	Reserved.	06_2EH
		26	HW_BLOCK: Set when DCA is blocked by HW (e.g. CR0.CD = 1).	06_2EH
		31:27	Reserved.	06_2EH
...				
38EH	910	IA32_PERF_GLOBAL_STATUS (MSR_PERF_GLOBAL_STATUS)	Global Performance Counter Status (RO)	If CPUID.0AH: EAX[7:0] > 0
		0	Ovf_PMC0: Overflow status of IA32_PMC0	If CPUID.0AH: EAX[7:0] > 0
		1	Ovf_PMC1: Overflow status of IA32_PMC1	If CPUID.0AH: EAX[7:0] > 0
		2	Ovf_PMC2: Overflow status of IA32_PMC2	06_2EH
		3	Ovf_PMC3: Overflow status of IA32_PMC3	06_2EH
		31:4	Reserved	
		32	Ovf_FixedCtr0: Overflow status of IA32_FIXED_CTR0	If CPUID.0AH: EAX[7:0] > 1
		33	Ovf_FixedCtr1: Overflow status of IA32_FIXED_CTR1	If CPUID.0AH: EAX[7:0] > 1
		34	Ovf_FixedCtr2: Overflow status of IA32_FIXED_CTR2	If CPUID.0AH: EAX[7:0] > 1
		60:35	Reserved	

		61	Ovf_Uncore: Uncore counter overflow status	06_2EH
		62	OvfBuf: DS SAVE area Buffer overflow status	If CPUID.0AH: EAX[7:0] > 0
		63	CondChg: status bits of this register has changed	If CPUID.0AH: EAX[7:0] > 0
...				
390H	912	IA32_PERF_GLOBAL_OVF_CTRL (MSR_PERF_GLOBAL_OVF_CTRL)	Global Performance Counter Overflow Control (R/W)	If CPUID.0AH: EAX[7:0] > 0
		0	Set 1 to Clear Ovf_PMC0 bit	If CPUID.0AH: EAX[7:0] > 0
		1	Set 1 to Clear Ovf_PMC1 bit	If CPUID.0AH: EAX[7:0] > 0
		31:2	Reserved	
		32	Set 1 to Clear Ovf_FIXED_CTR0 bit	If CPUID.0AH: EAX[7:0] > 1
		33	Set 1 to Clear Ovf_FIXED_CTR1 bit	If CPUID.0AH: EAX[7:0] > 1
		34	Set 1 to Clear Ovf_FIXED_CTR2 bit	If CPUID.0AH: EAX[7:0] > 1
		60:35	Reserved	
		61	Set 1 to Clear Ovf_Uncore: bit	06_2EH
		62	Set 1 to Clear OvfBuf: bit	If CPUID.0AH: EAX[7:0] > 0
		63	Set to 1to clear CondChg: bit	If CPUID.0AH: EAX[7:0] > 0
3F1H	1009	IA32_PEBS_ENABLE	PEBS Control (R/W)	
		0	Enable PEBS on IA32_PMC0	06_0FH
		1	Enable PEBS on IA32_PMC1	06_0FH
		2	Enable PEBS on IA32_PMC2	06_0FH
		3	Enable PEBS on IA32_PMC3	06_0FH
		31:4	Reserved	
		32	Enable Load Latency on IA32_PMC0	06_0FH
		33	Enable Load Latency on IA32_PMC1	06_0FH
		34	Enable Load Latency on IA32_PMC2	06_0FH
		35	Enable Load Latency on IA32_PMC3	06_0FH

		63:36	Reserved	
…				

…

B.4 MSRS IN THE INTEL® MICROARCHITECTURE CODENAME NEHALEM

Table B-5 lists model-specific registers (MSRs) that are common for Intel microarchitecture codename Nehalem. These include Intel Core i7 and i5 processor family. Architectural MSR addresses are also included in Table B-5. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_1AH, 06_1EH, 06_1FH, 06_2EH, see Table B-1. Additional MSRs specific to 06_1AH, 06_1EH, 06_1FH are listed in Table Table B-6.

…

B.4.1 Additional MSRs In the Intel® Xeon Processor 5500 and 3400 Series

Intel Xeon Processor 5500 and 3400 series support additional model-specific registers listed in Table Table B-6. These MSRs also apply to Intel Core i7 and i5 processor family CPUID signature with DisplayFamily_DisplayModel of 06_1AH, 06_1EH and 06_1FH, see Table B-1.

Table B-6 Additional MSRs in Intel Xeon Processor 5500 and 3400 Series

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Actual maximum turbo frequency is multiplied by 133.33MHz. (not available to model 06_2EH)
		7:0		**Maximum Turbo Ratio Limit 1C. (R/O)** maximum Turbo mode ratio limit with 1 core active.
		15:8		**Maximum Turbo Ratio Limit 2C. (R/O)** maximum Turbo mode ratio limit with 2cores active.
		23:16		**Maximum Turbo Ratio Limit 3C. (R/O)** maximum Turbo mode ratio limit with 3cores active.
		31:24		**Maximum Turbo Ratio Limit 4C. (R/O)** maximum Turbo mode ratio limit with 4 cores active.
		63:32		Reserved.
…				

…

B.4.2 Additional MSRs In the Intel® Xeon Processor 7500 Series

Intel Xeon Processor 7500 series support MSRs listed in Table B-5 and additional model-specific registers listed in Table Table B-7.

Table B-7 Additional MSRs in Intel Xeon Processor 7500 Series

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
...				
394H	816	MSR_W_PMON_FIXED_CTR	Package	Uncore W-box perfmon fixed counter
395H	817	MSR_W_PMON_FIXED_CTR_CTL	Package	Uncore U-box perfmon fixed counter control MSR
...				
C00H	3072	MSR_U_PMON_GLOBAL_CTRL	Package	Uncore U-box perfmon global control MSR
C01H	3073	MSR_U_PMON_GLOBAL_STATUS	Package	Uncore U-box perfmon global status MSR
C02H	3074	MSR_U_PMON_GLOBAL_OVF_CTRL	Package	Uncore U-box perfmon global overflow control MSR
C10H	3088	MSR_U_PMON_EVNT_SEL	Package	Uncore U-box perfmon event select MSR
C11H	3089	MSR_U_PMON_CTR	Package	Uncore U-box perfmon counter MSR
C20H	3104	MSR_B0_PMON_BOX_CTRL	Package	Uncore B-box 0 perfmon local box control MSR
C21H	3105	MSR_B0_PMON_BOX_STATUS	Package	Uncore B-box 0 perfmon local box status MSR
C22H	3106	MSR_B0_PMON_BOX_OVF_CTRL	Package	Uncore B-box 0 perfmon local box overflow control MSR
C30H	3120	MSR_B0_PMON_EVNT_SEL0	Package	Uncore B-box 0 perfmon event select MSR
C31H	3121	MSR_B0_PMON_CTR0	Package	Uncore B-box 0 perfmon counter MSR
C32H	3122	MSR_B0_PMON_EVNT_SEL1	Package	Uncore B-box 0 perfmon event select MSR
C33H	3123	MSR_B0_PMON_CTR1	Package	Uncore B-box 0 perfmon counter MSR
C34H	3124	MSR_B0_PMON_EVNT_SEL2	Package	Uncore B-box 0 perfmon event select MSR
C35H	3125	MSR_B0_PMON_CTR2	Package	Uncore B-box 0 perfmon counter MSR

C36H	3126	MSR_B0_PMON_EVNT_SEL3	Package	Uncore B-box 0 perfmon event select MSR
C37H	3127	MSR_B0_PMON_CTR3	Package	Uncore B-box 0 perfmon counter MSR
C40H	3136	MSR_S0_PMON_BOX_CTRL	Package	Uncore S-box 0 perfmon local box control MSR
C41H	3137	MSR_S0_PMON_BOX_STATUS	Package	Uncore S-box 0 perfmon local box status MSR
C42H	3138	MSR_S0_PMON_BOX_OVF_CTRL	Package	Uncore S-box 0 perfmon local box overflow control MSR
C50H	3152	MSR_S0_PMON_EVNT_SEL0	Package	Uncore S-box 0 perfmon event select MSR
C51H	3153	MSR_S0_PMON_CTR0	Package	Uncore S-box 0 perfmon counter MSR
C52H	3154	MSR_S0_PMON_EVNT_SEL1	Package	Uncore S-box 0 perfmon event select MSR
C53H	3155	MSR_S0_PMON_CTR1	Package	Uncore S-box 0 perfmon counter MSR
C54H	3156	MSR_S0_PMON_EVNT_SEL2	Package	Uncore S-box 0 perfmon event select MSR
C55H	3157	MSR_S0_PMON_CTR2	Package	Uncore S-box 0 perfmon counter MSR
C56H	3158	MSR_S0_PMON_EVNT_SEL3	Package	Uncore S-box 0 perfmon event select MSR
C57H	3159	MSR_S0_PMON_CTR3	Package	Uncore S-box 0 perfmon counter MSR
C60H	3168	MSR_B1_PMON_BOX_CTRL	Package	Uncore B-box 1 perfmon local box control MSR
C61H	3169	MSR_B1_PMON_BOX_STATUS	Package	Uncore B-box 1 perfmon local box status MSR
C62H	3170	MSR_B1_PMON_BOX_OVF_CTRL	Package	Uncore B-box 1 perfmon local box overflow control MSR
C70H	3184	MSR_B1_PMON_EVNT_SEL0	Package	Uncore B-box 1 perfmon event select MSR
C71H	3185	MSR_B1_PMON_CTR0	Package	Uncore B-box 1 perfmon counter MSR
C72H	3186	MSR_B1_PMON_EVNT_SEL1	Package	Uncore B-box 1 perfmon event select MSR
C73H	3187	MSR_B1_PMON_CTR1	Package	Uncore B-box 1 perfmon counter MSR
C74H	3188	MSR_B1_PMON_EVNT_SEL2	Package	Uncore B-box 1 perfmon event select MSR

C75H	3189	MSR_B1_PMON_CTR2	Package	Uncore B-box 1 perfmon counter MSR
C76H	3190	MSR_B1_PMON_EVNT_SEL3	Package	Uncore B-box 1vperfmon event select MSR
C77H	3191	MSR_B1_PMON_CTR3	Package	Uncore B-box 1 perfmon counter MSR
C80H	3120	MSR_W_PMON_BOX_CTRL	Package	Uncore W-box perfmon local box control MSR
C81H	3121	MSR_W_PMON_BOX_STATUS	Package	Uncore W-box perfmon local box status MSR
C82H	3122	MSR_W_PMON_BOX_OVF_CTRL	Package	Uncore W-box perfmon local box overflow control MSR
C90H	3136	MSR_W_PMON_EVNT_SEL0	Package	Uncore W-box perfmon event select MSR
C91H	3137	MSR_W_PMON_CTR0	Package	Uncore W-box perfmon counter MSR
C92H	3138	MSR_W_PMON_EVNT_SEL1	Package	Uncore W-box perfmon event select MSR
C93H	3139	MSR_W_PMON_CTR1	Package	Uncore W-box perfmon counter MSR
C94H	3140	MSR_W_PMON_EVNT_SEL2	Package	Uncore W-box perfmon event select MSR
C95H	3141	MSR_W_PMON_CTR2	Package	Uncore W-box perfmon counter MSR
C96H	3142	MSR_W_PMON_EVNT_SEL3	Package	Uncore W-box perfmon event select MSR
C97H	3143	MSR_W_PMON_CTR3	Package	Uncore W-box perfmon counter MSR
CA0H	3232	MSR_M0_PMON_BOX_CTRL	Package	Uncore M-box 0 perfmon local box control MSR
CA1H	3233	MSR_M0_PMON_BOX_STATUS	Package	Uncore M-box 0 perfmon local box status MSR
CA2H	3234	MSR_M0_PMON_BOX_OVF_CTRL	Package	Uncore M-box 0 perfmon local box overflow control MSR
CA4H	3236	MSR_M0_PMON_TIMESTAMP	Package	Uncore M-box 0 perfmon time stamp unit select MSR
CA5H	3237	MSR_M0_PMON_DSP	Package	Uncore M-box 0 perfmon DSP unit select MSR
CA6H	3238	MSR_M0_PMON_ISS	Package	Uncore M-box 0 perfmon ISS unit select MSR
CA7H	3239	MSR_M0_PMON_MAP	Package	Uncore M-box 0 perfmon MAP unit select MSR

CA8H	3240	MSR_M0_PMON_MSC_THR	Package	Uncore M-box 0 perfmon MIC THR select MSR
CA9H	3241	MSR_M0_PMON_PGT	Package	Uncore M-box 0 perfmon PGT unit select MSR
CAAH	3242	MSR_M0_PMON_PLD	Package	Uncore M-box 0 perfmon PLD unit select MSR
CABH	3243	MSR_M0_PMON_ZDP	Package	Uncore M-box 0 perfmon ZDP unit select MSR
CB0H	3248	MSR_M0_PMON_EVNT_SEL0	Package	Uncore M-box 0 perfmon event select MSR
CB1H	3249	MSR_M0_PMON_CTR0	Package	Uncore M-box 0 perfmon counter MSR
CB2H	3250	MSR_M0_PMON_EVNT_SEL1	Package	Uncore M-box 0 perfmon event select MSR
CB3H	3251	MSR_M0_PMON_CTR1	Package	Uncore M-box 0 perfmon counter MSR
CB4H	3252	MSR_M0_PMON_EVNT_SEL2	Package	Uncore M-box 0 perfmon event select MSR
CB5H	3253	MSR_M0_PMON_CTR2	Package	Uncore M-box 0 perfmon counter MSR
CB6H	3254	MSR_M0_PMON_EVNT_SEL3	Package	Uncore M-box 0 perfmon event select MSR
CB7H	3255	MSR_M0_PMON_CTR3	Package	Uncore M-box 0 perfmon counter MSR
CB8H	3256	MSR_M0_PMON_EVNT_SEL4	Package	Uncore M-box 0 perfmon event select MSR
CB9H	3257	MSR_M0_PMON_CTR4	Package	Uncore M-box 0 perfmon counter MSR
CBAH	3258	MSR_M0_PMON_EVNT_SEL5	Package	Uncore M-box 0 perfmon event select MSR
CBBH	3259	MSR_M0_PMON_CTR5	Package	Uncore M-box 0 perfmon counter MSR
CC0H	3264	MSR_S1_PMON_BOX_CTRL	Package	Uncore S-box 1 perfmon local box control MSR
CC1H	3265	MSR_S1_PMON_BOX_STATUS	Package	Uncore S-box 1 perfmon local box status MSR
CC2H	3266	MSR_S1_PMON_BOX_OVF_CTRL	Package	Uncore S-box 1 perfmon local box overflow control MSR
CD0H	3280	MSR_S1_PMON_EVNT_SEL0	Package	Uncore S-box 1 perfmon event select MSR
CD1H	3281	MSR_S1_PMON_CTR0	Package	Uncore S-box 1 perfmon counter MSR

CD2H	3282	MSR_S1_PMON_EVNT_SEL1	Package	Uncore S-box 1 perfmon event select MSR
CD3H	3283	MSR_S1_PMON_CTR1	Package	Uncore S-box 1 perfmon counter MSR
CD4H	3284	MSR_S1_PMON_EVNT_SEL2	Package	Uncore S-box 1 perfmon event select MSR
CD5H	3285	MSR_S1_PMON_CTR2	Package	Uncore S-box 1 perfmon counter MSR
CD6H	3286	MSR_S1_PMON_EVNT_SEL3	Package	Uncore S-box 1 perfmon event select MSR
CD7H	3287	MSR_S1_PMON_CTR3	Package	Uncore S-box 1 perfmon counter MSR
CE0H	3296	MSR_M1_PMON_BOX_CTRL	Package	Uncore M-box 1 perfmon local box control MSR
CE1H	3297	MSR_M1_PMON_BOX_STATUS	Package	Uncore M-box 1 perfmon local box status MSR
CE2H	3298	MSR_M1_PMON_BOX_OVF_CTRL	Package	Uncore M-box 1 perfmon local box overflow control MSR
CE4H	3300	MSR_M1_PMON_TIMESTAMP	Package	Uncore M-box 1 perfmon time stamp unit select MSR
CE5H	3301	MSR_M1_PMON_DSP	Package	Uncore M-box 1 perfmon DSP unit select MSR
CE6H	3302	MSR_M1_PMON_ISS	Package	Uncore M-box 1 perfmon ISS unit select MSR
CE7H	3303	MSR_M1_PMON_MAP	Package	Uncore M-box 1 perfmon MAP unit select MSR
CE8H	3304	MSR_M1_PMON_MSC_THR	Package	Uncore M-box 1 perfmon MIC THR select MSR
CE9H	3305	MSR_M1_PMON_PGT	Package	Uncore M-box 1 perfmon PGT unit select MSR
CEAH	3306	MSR_M1_PMON_PLD	Package	Uncore M-box 1 perfmon PLD unit select MSR
CEBH	3307	MSR_M1_PMON_ZDP	Package	Uncore M-box 1 perfmon ZDP unit select MSR
CF0H	3312	MSR_M1_PMON_EVNT_SEL0	Package	Uncore M-box 1 perfmon event select MSR
CF1H	3313	MSR_M1_PMON_CTR0	Package	Uncore M-box 1 perfmon counter MSR
CF2H	3314	MSR_M1_PMON_EVNT_SEL1	Package	Uncore M-box 1 perfmon event select MSR
CF3H	3315	MSR_M1_PMON_CTR1	Package	Uncore M-box 1 perfmon counter MSR

CF4H	3316	MSR_M1_PMON_EVNT_SEL2	Package	Uncore M-box 1 perfmon event select MSR
CF5H	3317	MSR_M1_PMON_CTR2	Package	Uncore M-box 1 perfmon counter MSR
CF6H	3318	MSR_M1_PMON_EVNT_SEL3	Package	Uncore M-box 1 perfmon event select MSR
CF7H	3319	MSR_M1_PMON_CTR3	Package	Uncore M-box 1 perfmon counter MSR
CF8H	3320	MSR_M1_PMON_EVNT_SEL4	Package	Uncore M-box 1 perfmon event select MSR
CF9H	3321	MSR_M1_PMON_CTR4	Package	Uncore M-box 1 perfmon counter MSR
CFAH	3322	MSR_M1_PMON_EVNT_SEL5	Package	Uncore M-box 1 perfmon event select MSR
CFBH	3323	MSR_M1_PMON_CTR5	Package	Uncore M-box 1 perfmon counter MSR
D00H	3328	MSR_C0_PMON_BOX_CTRL	Package	Uncore C-box 0 perfmon local box control MSR
D01H	3329	MSR_C0_PMON_BOX_STATUS	Package	Uncore C-box 0 perfmon local box status MSR
D02H	3330	MSR_C0_PMON_BOX_OVF_CTRL	Package	Uncore C-box 0 perfmon local box overflow control MSR
D10H	3344	MSR_C0_PMON_EVNT_SEL0	Package	Uncore C-box 0 perfmon event select MSR
D11H	3345	MSR_C0_PMON_CTR0	Package	Uncore C-box 0 perfmon counter MSR
D12H	3346	MSR_C0_PMON_EVNT_SEL1	Package	Uncore C-box 0 perfmon event select MSR
D13H	3347	MSR_C0_PMON_CTR1	Package	Uncore C-box 0 perfmon counter MSR
D14H	3348	MSR_C0_PMON_EVNT_SEL2	Package	Uncore C-box 0 perfmon event select MSR
D15H	3349	MSR_C0_PMON_CTR2	Package	Uncore C-box 0 perfmon counter MSR
D16H	3350	MSR_C0_PMON_EVNT_SEL3	Package	Uncore C-box 0 perfmon event select MSR
D17H	3351	MSR_C0_PMON_CTR3	Package	Uncore C-box 0 perfmon counter MSR
D18H	3352	MSR_C0_PMON_EVNT_SEL4	Package	Uncore C-box 0 perfmon event select MSR
D19H	3353	MSR_C0_PMON_CTR4	Package	Uncore C-box 0 perfmon counter MSR

D1AH	3354	MSR_C0_PMON_EVNT_SEL5	Package	Uncore C-box 0 perfmon event select MSR
D1BH	3355	MSR_C0_PMON_CTR5	Package	Uncore C-box 0 perfmon counter MSR
D20H	3360	MSR_C4_PMON_BOX_CTRL	Package	Uncore C-box 4 perfmon local box control MSR
D21H	3361	MSR_C4_PMON_BOX_STATUS	Package	Uncore C-box 4 perfmon local box status MSR
D22H	3362	MSR_C4_PMON_BOX_OVF_CTRL	Package	Uncore C-box 4 perfmon local box overflow control MSR
D30H	3376	MSR_C4_PMON_EVNT_SEL0	Package	Uncore C-box 4 perfmon event select MSR
D31H	3377	MSR_C4_PMON_CTR0	Package	Uncore C-box 4 perfmon counter MSR
D32H	3378	MSR_C4_PMON_EVNT_SEL1	Package	Uncore C-box 4 perfmon event select MSR
D33H	3379	MSR_C4_PMON_CTR1	Package	Uncore C-box 4 perfmon counter MSR
D34H	3380	MSR_C4_PMON_EVNT_SEL2	Package	Uncore C-box 4 perfmon event select MSR
D35H	3381	MSR_C4_PMON_CTR2	Package	Uncore C-box 4 perfmon counter MSR
D36H	3382	MSR_C4_PMON_EVNT_SEL3	Package	Uncore C-box 4 perfmon event select MSR
D37H	3383	MSR_C4_PMON_CTR3	Package	Uncore C-box 4 perfmon counter MSR
D38H	3384	MSR_C4_PMON_EVNT_SEL4	Package	Uncore C-box 4 perfmon event select MSR
D39H	3385	MSR_C4_PMON_CTR4	Package	Uncore C-box 4 perfmon counter MSR
D3AH	3386	MSR_C4_PMON_EVNT_SEL5	Package	Uncore C-box 4 perfmon event select MSR
D3BH	3387	MSR_C4_PMON_CTR5	Package	Uncore C-box 4 perfmon counter MSR
D40H	3392	MSR_C2_PMON_BOX_CTRL	Package	Uncore C-box 2 perfmon local box control MSR
D41H	3393	MSR_C2_PMON_BOX_STATUS	Package	Uncore C-box 2 perfmon local box status MSR
D42H	3394	MSR_C2_PMON_BOX_OVF_CTRL	Package	Uncore C-box 2 perfmon local box overflow control MSR
D50H	3408	MSR_C2_PMON_EVNT_SEL0	Package	Uncore C-box 2 perfmon event select MSR

D51H	3409	MSR_C2_PMON_CTR0	Package	Uncore C-box 2 perfmon counter MSR
D52H	3410	MSR_C2_PMON_EVNT_SEL1	Package	Uncore C-box 2 perfmon event select MSR
D53H	3411	MSR_C2_PMON_CTR1	Package	Uncore C-box 2 perfmon counter MSR
D54H	3412	MSR_C2_PMON_EVNT_SEL2	Package	Uncore C-box 2 perfmon event select MSR
D55H	3413	MSR_C2_PMON_CTR2	Package	Uncore C-box 2 perfmon counter MSR
D56H	3414	MSR_C2_PMON_EVNT_SEL3	Package	Uncore C-box 2 perfmon event select MSR
D57H	3415	MSR_C2_PMON_CTR3	Package	Uncore C-box 2 perfmon counter MSR
D58H	3416	MSR_C2_PMON_EVNT_SEL4	Package	Uncore C-box 2 perfmon event select MSR
D59H	3417	MSR_C2_PMON_CTR4	Package	Uncore C-box 2 perfmon counter MSR
D5AH	3418	MSR_C2_PMON_EVNT_SEL5	Package	Uncore C-box 2 perfmon event select MSR
D5BH	3419	MSR_C2_PMON_CTR5	Package	Uncore C-box 2 perfmon counter MSR
D60H	3424	MSR_C6_PMON_BOX_CTRL	Package	Uncore C-box 6 perfmon local box control MSR
D61H	3425	MSR_C6_PMON_BOX_STATUS	Package	Uncore C-box 6 perfmon local box status MSR
D62H	3426	MSR_C6_PMON_BOX_OVF_CTRL	Package	Uncore C-box 6 perfmon local box overflow control MSR
D70H	3440	MSR_C6_PMON_EVNT_SEL0	Package	Uncore C-box 6 perfmon event select MSR
D71H	3441	MSR_C6_PMON_CTR0	Package	Uncore C-box 6 perfmon counter MSR
D72H	3442	MSR_C6_PMON_EVNT_SEL1	Package	Uncore C-box 6 perfmon event select MSR
D73H	3443	MSR_C6_PMON_CTR1	Package	Uncore C-box 6 perfmon counter MSR
D74H	3444	MSR_C6_PMON_EVNT_SEL2	Package	Uncore C-box 6 perfmon event select MSR
D75H	3445	MSR_C6_PMON_CTR2	Package	Uncore C-box 6 perfmon counter MSR
D76H	3446	MSR_C6_PMON_EVNT_SEL3	Package	Uncore C-box 6 perfmon event select MSR

D77H	3447	MSR_C6_PMON_CTR3	Package	Uncore C-box 6 perfmon counter MSR
D78H	3448	MSR_C6_PMON_EVNT_SEL4	Package	Uncore C-box 6 perfmon event select MSR
D79H	3449	MSR_C6_PMON_CTR4	Package	Uncore C-box 6 perfmon counter MSR
D7AH	3450	MSR_C6_PMON_EVNT_SEL5	Package	Uncore C-box 6 perfmon event select MSR
D7BH	3451	MSR_C6_PMON_CTR5	Package	Uncore C-box 6 perfmon counter MSR
D80H	3456	MSR_C1_PMON_BOX_CTRL	Package	Uncore C-box 1 perfmon local box control MSR
D81H	3457	MSR_C1_PMON_BOX_STATUS	Package	Uncore C-box 1 perfmon local box status MSR
D82H	3458	MSR_C1_PMON_BOX_OVF_CTRL	Package	Uncore C-box 1 perfmon local box overflow control MSR
D90H	3472	MSR_C1_PMON_EVNT_SEL0	Package	Uncore C-box 1 perfmon event select MSR
D91H	3473	MSR_C1_PMON_CTR0	Package	Uncore C-box 1 perfmon counter MSR
D92H	3474	MSR_C1_PMON_EVNT_SEL1	Package	Uncore C-box 1 perfmon event select MSR
D93H	3475	MSR_C1_PMON_CTR1	Package	Uncore C-box 1 perfmon counter MSR
D94H	3476	MSR_C1_PMON_EVNT_SEL2	Package	Uncore C-box 1 perfmon event select MSR
D95H	3477	MSR_C1_PMON_CTR2	Package	Uncore C-box 1 perfmon counter MSR
D96H	3478	MSR_C1_PMON_EVNT_SEL3	Package	Uncore C-box 1 perfmon event select MSR
D97H	3479	MSR_C1_PMON_CTR3	Package	Uncore C-box 1 perfmon counter MSR
D98H	3480	MSR_C1_PMON_EVNT_SEL4	Package	Uncore C-box 1 perfmon event select MSR
D99H	3481	MSR_C1_PMON_CTR4	Package	Uncore C-box 1 perfmon counter MSR
D9AH	3482	MSR_C1_PMON_EVNT_SEL5	Package	Uncore C-box 1 perfmon event select MSR
D9BH	3483	MSR_C1_PMON_CTR5	Package	Uncore C-box 1 perfmon counter MSR
DA0H	3488	MSR_C5_PMON_BOX_CTRL	Package	Uncore C-box 5 perfmon local box control MSR

DA1H	3489	MSR_C5_PMON_BOX_STATUS	Package	Uncore C-box 5 perfmon local box status MSR
DA2H	3490	MSR_C5_PMON_BOX_OVF_CTRL	Package	Uncore C-box 5 perfmon local box overflow control MSR
DB0H	3504	MSR_C5_PMON_EVNT_SEL0	Package	Uncore C-box 5 perfmon event select MSR
DB1H	3505	MSR_C5_PMON_CTR0	Package	Uncore C-box 5 perfmon counter MSR
DB2H	3506	MSR_C5_PMON_EVNT_SEL1	Package	Uncore C-box 5 perfmon event select MSR
DB3H	3507	MSR_C5_PMON_CTR1	Package	Uncore C-box 5 perfmon counter MSR
DB4H	3508	MSR_C5_PMON_EVNT_SEL2	Package	Uncore C-box 5 perfmon event select MSR
DB5H	3509	MSR_C5_PMON_CTR2	Package	Uncore C-box 5 perfmon counter MSR
DB6H	3510	MSR_C5_PMON_EVNT_SEL3	Package	Uncore C-box 5 perfmon event select MSR
DB7H	3511	MSR_C5_PMON_CTR3	Package	Uncore C-box 5 perfmon counter MSR
DB8H	3512	MSR_C5_PMON_EVNT_SEL4	Package	Uncore C-box 5 perfmon event select MSR
DB9H	3513	MSR_C5_PMON_CTR4	Package	Uncore C-box 5 perfmon counter MSR
DBAH	3514	MSR_C5_PMON_EVNT_SEL5	Package	Uncore C-box 5 perfmon event select MSR
DBBH	3515	MSR_C5_PMON_CTR5	Package	Uncore C-box 5 perfmon counter MSR
DC0H	3520	MSR_C3_PMON_BOX_CTRL	Package	Uncore C-box 3 perfmon local box control MSR
DC1H	3521	MSR_C3_PMON_BOX_STATUS	Package	Uncore C-box 3 perfmon local box status MSR
DC2H	3522	MSR_C3_PMON_BOX_OVF_CTRL	Package	Uncore C-box 3 perfmon local box overflow control MSR
DD0H	3536	MSR_C3_PMON_EVNT_SEL0	Package	Uncore C-box 3 perfmon event select MSR
DD1H	3537	MSR_C3_PMON_CTR0	Package	Uncore C-box 3 perfmon counter MSR
DD2H	3538	MSR_C3_PMON_EVNT_SEL1	Package	Uncore C-box 3 perfmon event select MSR
DD3H	3539	MSR_C3_PMON_CTR1	Package	Uncore C-box 3 perfmon counter MSR

DD4H	3540	MSR_C3_PMON_EVNT_SEL2	Package	Uncore C-box 3 perfmon event select MSR
DD5H	3541	MSR_C3_PMON_CTR2	Package	Uncore C-box 3 perfmon counter MSR
DD6H	3542	MSR_C3_PMON_EVNT_SEL3	Package	Uncore C-box 3 perfmon event select MSR
DD7H	3543	MSR_C3_PMON_CTR3	Package	Uncore C-box 3 perfmon counter MSR
DD8H	3544	MSR_C3_PMON_EVNT_SEL4	Package	Uncore C-box 3 perfmon event select MSR
DD9H	3545	MSR_C3_PMON_CTR4	Package	Uncore C-box 3 perfmon counter MSR
DDAH	3546	MSR_C3_PMON_EVNT_SEL5	Package	Uncore C-box 3 perfmon event select MSR
DDBH	3547	MSR_C3_PMON_CTR5	Package	Uncore C-box 3 perfmon counter MSR
DE0H	3552	MSR_C7_PMON_BOX_CTRL	Package	Uncore C-box 7 perfmon local box control MSR
DE1H	3553	MSR_C7_PMON_BOX_STATUS	Package	Uncore C-box 7 perfmon local box status MSR
DE2H	3554	MSR_C7_PMON_BOX_OVF_CTRL	Package	Uncore C-box 7 perfmon local box overflow control MSR
DF0H	3568	MSR_C7_PMON_EVNT_SEL0	Package	Uncore C-box 7 perfmon event select MSR
DF1H	3569	MSR_C7_PMON_CTR0	Package	Uncore C-box 7 perfmon counter MSR
DF2H	3570	MSR_C7_PMON_EVNT_SEL1	Package	Uncore C-box 7 perfmon event select MSR
DF3H	3571	MSR_C7_PMON_CTR1	Package	Uncore C-box 7 perfmon counter MSR
DF4H	3572	MSR_C7_PMON_EVNT_SEL2	Package	Uncore C-box 7 perfmon event select MSR
DF5H	3573	MSR_C7_PMON_CTR2	Package	Uncore C-box 7 perfmon counter MSR
DF6H	3574	MSR_C7_PMON_EVNT_SEL3	Package	Uncore C-box 7 perfmon event select MSR
DF7H	3575	MSR_C7_PMON_CTR3	Package	Uncore C-box 7 perfmon counter MSR
DF8H	3576	MSR_C7_PMON_EVNT_SEL4	Package	Uncore C-box 7 perfmon event select MSR
DF9H	3577	MSR_C7_PMON_CTR4	Package	Uncore C-box 7 perfmon counter MSR

DFAH	3578	MSR_C7_PMON_EVNT_SEL5	Package	Uncore C-box 7 perfmon event select MSR
DFBH	3579	MSR_C7_PMON_CTR5	Package	Uncore C-box 7 perfmon counter MSR
E00H	3584	MSR_R0_PMON_BOX_CTRL	Package	Uncore R-box 0 perfmon local box control MSR
E01H	3585	MSR_R0_PMON_BOX_STATUS	Package	Uncore R-box 0 perfmon local box status MSR
E02H	3586	MSR_R0_PMON_BOX_OVF_CTRL	Package	Uncore R-box 0 perfmon local box overflow control MSR
E04H	3588	MSR_R0_PMON_IPERF0_P0	Package	Uncore R-box 0 perfmon IPERF0 unit Port 0 select MSR
E05H	3589	MSR_R0_PMON_IPERF0_P1	Package	Uncore R-box 0 perfmon IPERF0 unit Port 1 select MSR
E06H	3590	MSR_R0_PMON_IPERF0_P2	Package	Uncore R-box 0 perfmon IPERF0 unit Port 2 select MSR
E07H	3591	MSR_R0_PMON_IPERF0_P3	Package	Uncore R-box 0 perfmon IPERF0 unit Port 3 select MSR
E08H	3592	MSR_R0_PMON_IPERF0_P4	Package	Uncore R-box 0 perfmon IPERF0 unit Port 4 select MSR
E09H	3593	MSR_R0_PMON_IPERF0_P5	Package	Uncore R-box 0 perfmon IPERF0 unit Port 5 select MSR
E0AH	3594	MSR_R0_PMON_IPERF0_P6	Package	Uncore R-box 0 perfmon IPERF0 unit Port 6 select MSR
E0BH	3595	MSR_R0_PMON_IPERF0_P7	Package	Uncore R-box 0 perfmon IPERF0 unit Port 7 select MSR
E0CH	3596	MSR_R0_PMON_QLX_P0	Package	Uncore R-box 0 perfmon QLX unit Port 0 select MSR
E0DH	3597	MSR_R0_PMON_QLX_P1	Package	Uncore R-box 0 perfmon QLX unit Port 1 select MSR
E0EH	3598	MSR_R0_PMON_QLX_P2	Package	Uncore R-box 0 perfmon QLX unit Port 2 select MSR
E0FH	3599	MSR_R0_PMON_QLX_P3	Package	Uncore R-box 0 perfmon QLX unit Port 3 select MSR
E10H	3600	MSR_R0_PMON_EVNT_SEL0	Package	Uncore R-box 0 perfmon event select MSR
E11H	3601	MSR_R0_PMON_CTR0	Package	Uncore R-box 0 perfmon counter MSR
E12H	3602	MSR_R0_PMON_EVNT_SEL1	Package	Uncore R-box 0 perfmon event select MSR
E13H	3603	MSR_R0_PMON_CTR1	Package	Uncore R-box 0 perfmon counter MSR

E14H	3604	MSR_R0_PMON_EVNT_SEL2	Package	Uncore R-box 0 perfmon event select MSR
E15H	3605	MSR_R0_PMON_CTR2	Package	Uncore R-box 0 perfmon counter MSR
E16H	3606	MSR_R0_PMON_EVNT_SEL3	Package	Uncore R-box 0 perfmon event select MSR
E17H	3607	MSR_R0_PMON_CTR3	Package	Uncore R-box 0 perfmon counter MSR
E18H	3608	MSR_R0_PMON_EVNT_SEL4	Package	Uncore R-box 0 perfmon event select MSR
E19H	3609	MSR_R0_PMON_CTR4	Package	Uncore R-box 0 perfmon counter MSR
E1AH	3610	MSR_R0_PMON_EVNT_SEL5	Package	Uncore R-box 0 perfmon event select MSR
E1BH	3611	MSR_R0_PMON_CTR5	Package	Uncore R-box 0 perfmon counter MSR
E1CH	3612	MSR_R0_PMON_EVNT_SEL6	Package	Uncore R-box 0 perfmon event select MSR
E1DH	3613	MSR_R0_PMON_CTR6	Package	Uncore R-box 0 perfmon counter MSR
E1EH	3614	MSR_R0_PMON_EVNT_SEL7	Package	Uncore R-box 0 perfmon event select MSR
E1FH	3615	MSR_R0_PMON_CTR7	Package	Uncore R-box 0 perfmon counter MSR
E20H	3616	MSR_R1_PMON_BOX_CTRL	Package	Uncore R-box 1 perfmon local box control MSR
E21H	3617	MSR_R1_PMON_BOX_STATUS	Package	Uncore R-box 1 perfmon local box status MSR
E22H	3618	MSR_R1_PMON_BOX_OVF_CTRL	Package	Uncore R-box 1 perfmon local box overflow control MSR
E24H	3620	MSR_R1_PMON_IPERF1_P8	Package	Uncore R-box 1 perfmon IPERF1 unit Port 8 select MSR
E25H	3621	MSR_R1_PMON_IPERF1_P9	Package	Uncore R-box 1 perfmon IPERF1 unit Port 9 select MSR
E26H	3622	MSR_R1_PMON_IPERF1_P10	Package	Uncore R-box 1 perfmon IPERF1 unit Port 10 select MSR
E27H	3623	MSR_R1_PMON_IPERF1_P11	Package	Uncore R-box 1 perfmon IPERF1 unit Port 11 select MSR
E28H	3624	MSR_R1_PMON_IPERF1_P12	Package	Uncore R-box 1 perfmon IPERF1 unit Port 12 select MSR
E29H	3625	MSR_R1_PMON_IPERF1_P13	Package	Uncore R-box 1 perfmon IPERF1 unit Port 13 select MSR

E2AH	3626	MSR_R1_PMON_IPERF1_P14	Package	Uncore R-box 1 perfmon IPERF1 unit Port 14 select MSR
E2BH	3627	MSR_R1_PMON_IPERF1_P15	Package	Uncore R-box 1 perfmon IPERF1 unit Port 15 select MSR
E2CH	3628	MSR_R1_PMON_QLX_P4	Package	Uncore R-box 1 perfmon QLX unit Port 4 select MSR
E2DH	3629	MSR_R1_PMON_QLX_P5	Package	Uncore R-box 1 perfmon QLX unit Port 5 select MSR
E2EH	3630	MSR_R1_PMON_QLX_P6	Package	Uncore R-box 1 perfmon QLX unit Port 6 select MSR
E2FH	3631	MSR_R1_PMON_QLX_P7	Package	Uncore R-box 1 perfmon QLX unit Port 7 select MSR
E30H	3632	MSR_R1_PMON_EVNT_SEL8	Package	Uncore R-box 1 perfmon event select MSR
E31H	3633	MSR_R1_PMON_CTR8	Package	Uncore R-box 1 perfmon counter MSR
E32H	3634	MSR_R1_PMON_EVNT_SEL9	Package	Uncore R-box 1 perfmon event select MSR
E33H	3635	MSR_R1_PMON_CTR9	Package	Uncore R-box 1 perfmon counter MSR
E34H	3636	MSR_R1_PMON_EVNT_SEL10	Package	Uncore R-box 1 perfmon event select MSR
E35H	3637	MSR_R1_PMON_CTR10	Package	Uncore R-box 1 perfmon counter MSR
E36H	3638	MSR_R1_PMON_EVNT_SEL11	Package	Uncore R-box 1 perfmon event select MSR
E37H	3639	MSR_R1_PMON_CTR11	Package	Uncore R-box 1 perfmon counter MSR
E38H	3640	MSR_R1_PMON_EVNT_SEL12	Package	Uncore R-box 1 perfmon event select MSR
E39H	3641	MSR_R1_PMON_CTR12	Package	Uncore R-box 1 perfmon counter MSR
E3AH	3642	MSR_R1_PMON_EVNT_SEL13	Package	Uncore R-box 1 perfmon event select MSR
E3BH	3643	MSR_R1_PMON_CTR13	Package	Uncore R-box 1perfmon counter MSR
E3CH	3644	MSR_R1_PMON_EVNT_SEL14	Package	Uncore R-box 1 perfmon event select MSR
E3DH	3645	MSR_R1_PMON_CTR14	Package	Uncore R-box 1 perfmon counter MSR
E3EH	3646	MSR_R1_PMON_EVNT_SEL15	Package	Uncore R-box 1 perfmon event select MSR

E3FH	3647	MSR_R1_PMON_CTR15	Package	Uncore R-box 1 perfmon counter MSR
E45H	3653	MSR_B0_PMON_MATCH	Package	Uncore B-box 0 perfmon local box match MSR
E46H	3654	MSR_B0_PMON_MASK	Package	Uncore B-box 0 perfmon local box mask MSR
E49H	3657	MSR_S0_PMON_MATCH	Package	Uncore S-box 0 perfmon local box match MSR
E4AH	3658	MSR_S0_PMON_MASK	Package	Uncore S-box 0 perfmon local box mask MSR
E4DH	3661	MSR_B1_PMON_MATCH	Package	Uncore B-box 1 perfmon local box match MSR
E4EH	3662	MSR_B1_PMON_MASK	Package	Uncore B-box 1 perfmon local box mask MSR
E54H	3668	MSR_M0_PMON_MM_CONFIG	Package	Uncore M-box 0 perfmon local box address match/mask config MSR
E55H	3669	MSR_M0_PMON_ADDR_MATCH	Package	Uncore M-box 0 perfmon local box address match MSR
E56H	3670	MSR_M0_PMON_ADDR_MASK	Package	Uncore M-box 0 perfmon local box address mask MSR
E59H	3673	MSR_S1_PMON_MATCH	Package	Uncore S-box 1 perfmon local box match MSR
E5AH	3674	MSR_S1_PMON_MASK	Package	Uncore S-box 1 perfmon local box mask MSR
E5CH	3676	MSR_M1_PMON_MM_CONFIG	Package	Uncore M-box 1 perfmon local box address match/mask config MSR
E5DH	3677	MSR_M1_PMON_ADDR_MATCH	Package	Uncore M-box 1 perfmon local box address match MSR
E5EH	3678	MSR_M1_PMON_ADDR_MASK	Package	Uncore M-box 1 perfmon local box address mask MSR
3B5H	965	MSR_UNCORE_PMC5	Package	See Section 30.6.2.2, "Uncore Performance Event Configuration Facility."

B.5 MSRS IN THE INTEL® XEON PROCESSOR 5600 SERIES (INTEL MICROARCHITECTURE CODENAME WESTMERE)

Intel Xeon processor 5600 series (Intel® microarchitecture codename Westmere) supports the MSR interfaces listed in Table B-5, Table Table B-6, plus additional MSR listed in Table B-8. These MSRs also apply to Intel Core i7, i5 and i3 processor family with CPUID signature DisplayFamily_DisplayModel of 06_25H and 06_2CH, see Table B-1.

...

24. Updates to Appendix E, Volume 3B

Change bars show changes to Appendix E of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

…

Table E-2 Incremental Decoding Information: Processor Family 06H Machine Error Codes For Machine Check

Type	Bit No.	Bit Function	Bit Description
…			
	38	Timeout BINIT	This bit is asserted in IA32_MC*i*_STATUS if this component has experienced a ROB time-out, which indicates that no micro-instruction has been retired for a predetermined period of time. A ROB time-out occurs when the 15-bit ROB time-out counter carries a 1 out of its high order bit. [1] The timer is cleared when a micro-instruction retires, an exception is detected by the core processor, RESET is asserted, or when a ROB BINIT occurs.
…			

NOTES:

1. For processors with a CPUID signature of 06_0EH, a ROB time-out occurs when the 23-bit ROB time-out counter carries a 1 out of its high order bit.

…

Table E-4 Incremental Bus Error Codes of Machine Check for Processors Based on Intel Core Microarchitecture

Type	Bit No.	Bit Function	Bit Description
…			
	38	Timeout BINIT	This bit is asserted in IA32_MC*i*_STATUS if this component has experienced a ROB time-out, which indicates that no micro-instruction has been retired for a predetermined period of time. A ROB time-out occurs when the 23-bit ROB time-out counter carries a 1 out of its high order bit. The timer is cleared when a micro-instruction retires, an exception is detected by the core processor, RESET is asserted, or when a ROB BINIT occurs.
…			

…