

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

Documentation Changes

March 2012

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: 252046-035



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

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

Intel, the Intel logo, Pentium, Xeon, Intel NetBurst, Intel Core, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo, Intel Core 2 Extreme, Intel Pentium D, Itanium, Intel SpeedStep, MMX, Intel Atom, and VTune are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copyright © 1997-2012 Intel Corporation. All rights reserved.



Contents

Revision History	4
Preface	7
Summary Tables of Changes	8
Documentation Changes	9



Revision History

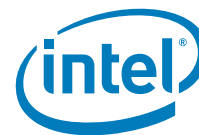
Revision	Description	Date
-001	<ul style="list-style-type: none">Initial release	November 2002
-002	<ul style="list-style-type: none">Added 1-10 Documentation Changes.Removed old Documentation Changes items that already have been incorporated in the published Software Developer's manual	December 2002
-003	<ul style="list-style-type: none">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	<ul style="list-style-type: none">Removed Documentation changes 1-17.Added Documentation changes 1-24.	June 2003
-005	<ul style="list-style-type: none">Removed Documentation Changes 1-24.Added Documentation Changes 1-15.	September 2003
-006	<ul style="list-style-type: none">Added Documentation Changes 16- 34.	November 2003
-007	<ul style="list-style-type: none">Updated Documentation changes 14, 16, 17, and 28.Added Documentation Changes 35-45.	January 2004
-008	<ul style="list-style-type: none">Removed Documentation Changes 1-45.Added Documentation Changes 1-5.	March 2004
-009	<ul style="list-style-type: none">Added Documentation Changes 7-27.	May 2004
-010	<ul style="list-style-type: none">Removed Documentation Changes 1-27.Added Documentation Changes 1.	August 2004
-011	<ul style="list-style-type: none">Added Documentation Changes 2-28.	November 2004
-012	<ul style="list-style-type: none">Removed Documentation Changes 1-28.Added Documentation Changes 1-16.	March 2005
-013	<ul style="list-style-type: none">Updated title.There are no Documentation Changes for this revision of the document.	July 2005
-014	<ul style="list-style-type: none">Added Documentation Changes 1-21.	September 2005
-015	<ul style="list-style-type: none">Removed Documentation Changes 1-21.Added Documentation Changes 1-20.	March 9, 2006
-016	<ul style="list-style-type: none">Added Documentation changes 21-23.	March 27, 2006
-017	<ul style="list-style-type: none">Removed Documentation Changes 1-23.Added Documentation Changes 1-36.	September 2006
-018	<ul style="list-style-type: none">Added Documentation Changes 37-42.	October 2006
-019	<ul style="list-style-type: none">Removed Documentation Changes 1-42.Added Documentation Changes 1-19.	March 2007
-020	<ul style="list-style-type: none">Added Documentation Changes 20-27.	May 2007
-021	<ul style="list-style-type: none">Removed Documentation Changes 1-27.Added Documentation Changes 1-6	November 2007
-022	<ul style="list-style-type: none">Removed Documentation Changes 1-6Added Documentation Changes 1-6	August 2008
-023	<ul style="list-style-type: none">Removed Documentation Changes 1-6Added Documentation Changes 1-21	March 2009



Revision	Description	Date
-024	<ul style="list-style-type: none"> Removed Documentation Changes 1-21 Added Documentation Changes 1-16 	June 2009
-025	<ul style="list-style-type: none"> Removed Documentation Changes 1-16 Added Documentation Changes 1-18 	September 2009
-026	<ul style="list-style-type: none"> Removed Documentation Changes 1-18 Added Documentation Changes 1-15 	December 2009
-027	<ul style="list-style-type: none"> Removed Documentation Changes 1-15 Added Documentation Changes 1-24 	March 2010
-028	<ul style="list-style-type: none"> Removed Documentation Changes 1-24 Added Documentation Changes 1-29 	June 2010
-029	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Added Documentation Changes 1-29 	September 2010
-030	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Added Documentation Changes 1-29 	January 2011
-031	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Added Documentation Changes 1-29 	April 2011
-032	<ul style="list-style-type: none"> Removed Documentation Changes 1-29 Added Documentation Changes 1-14 	May 2011
-033	<ul style="list-style-type: none"> Removed Documentation Changes 1-14 Added Documentation Changes 1-38 	October 2011
-034	<ul style="list-style-type: none"> Removed Documentation Changes 1-38 Added Documentation Changes 1-16 	December 2011
-035	<ul style="list-style-type: none"> Removed Documentation Changes 1-16 Added Documentation Changes 1-18 	March 2012

§





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
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture</i>	253665
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-M</i>	253666
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, N-Z</i>	253667
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C: Instruction Set Reference</i>	326018
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1</i>	253668
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2</i>	253669
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3</i>	326019

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 1, Volume 1
2	Updates to Chapter 1, Volume 2A
3	Updates to Chapter 3, Volume 2A
4	Updates to Chapter 4, Volume 2B
5	Updates to Appendix A, Volume 2C
6	Updates to Appendix B, Volume 2C
7	Updates to Chapter 1, Volume 3A
8	Updates to Chapter 4, Volume 3A
9	Updates to Chapter 10, Volume 3A
10	Updates to Chapter 14, Volume 3B
11	Updates to Chapter 17, Volume 3B
12	Updates to Chapter 18, Volume 3B
13	Updates to Chapter 19, Volume 3B
14	Updates to Chapter 25, Volume 3C
15	Updates to Chapter 26, Volume 3C
16	Updates to Chapter 27, Volume 3C
17	Update to Volume 3C
18	Updates to Chapter 33, Volume 3C



Documentation Changes

1. Updates to Chapter 1, Volume 1

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

...

1.1 INTEL® 64 AND IA-32 PROCESSORS COVERED IN THIS MANUAL

This manual set includes information pertaining primarily to the most recent Intel 64 and IA-32 processors, which include:

- Pentium® processors
- P6 family processors
- Pentium® 4 processors
- Pentium® M processors
- Intel® Xeon® processors
- Pentium® D processors
- Pentium® processor Extreme Editions
- 64-bit Intel® Xeon® processors
- Intel® Core™ Duo processor
- Intel® Core™ Solo processor
- Dual-Core Intel® Xeon® processor LV
- Intel® Core™2 Duo processor
- Intel® Core™2 Quad processor Q6000 series
- Intel® Xeon® processor 3000, 3200 series
- Intel® Xeon® processor 5000 series
- Intel® Xeon® processor 5100, 5300 series
- Intel® Core™2 Extreme processor X7000 and X6800 series
- Intel® Core™2 Extreme processor QX6000 series
- Intel® Xeon® processor 7100 series
- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® Xeon® processor 5200, 5400, 7400 series
- Intel® Core™2 Extreme processor QX9000 and X9000 series
- Intel® Core™2 Quad processor Q9000 series
- Intel® Core™2 Duo processor E8000, T9000 series
- Intel® Atom™ processor family



- Intel® Core™ i7 processor
- Intel® Core™ i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Xeon® processor E5 family
- Intel® Xeon® processor E3 family
- Intel® Core™ i7-3930K processor
- 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series

P6 family processors are IA-32 processors based on the P6 family microarchitecture. This includes the Pentium® Pro, Pentium® II, Pentium® III, and Pentium® III Xeon® processors.

The Pentium® 4, Pentium® D, and Pentium® processor Extreme Editions are based on the Intel NetBurst® microarchitecture. Most early Intel® Xeon® processors are based on the Intel NetBurst® microarchitecture. Intel Xeon processor 5000, 7100 series are based on the Intel NetBurst® microarchitecture.

The Intel® Core™ Duo, Intel® Core™ Solo and dual-core Intel® Xeon® processor LV are based on an improved Pentium® M processor microarchitecture.

The Intel® Xeon® processor 3000, 3200, 5100, 5300, 7200 and 7300 series, Intel® Pentium® dual-core, Intel® Core™2 Duo, Intel® Core™2 Quad, and Intel® Core™2 Extreme processors are based on Intel® Core™ microarchitecture.

The Intel® Xeon® processor 5200, 5400, 7400 series, Intel® Core™2 Quad processor Q9000 series, and Intel® Core™2 Extreme processor QX9000, X9000 series, Intel® Core™2 processor E8000 series are based on Enhanced Intel® Core™ microarchitecture.

The Intel® Atom™ processor family is based on the Intel® Atom™ microarchitecture and supports Intel 64 architecture.

The Intel® Core™ i7 processor and the Intel® Core™ i5 processor are based on the Intel® microarchitecture code name Nehalem and support Intel 64 architecture.

Processors based on Intel® microarchitecture code name Westmere support Intel 64 architecture.

P6 family, Pentium® M, Intel® Core™ Solo, Intel® Core™ Duo processors, dual-core Intel® Xeon® processor LV, and early generations of Pentium 4 and Intel Xeon processors support IA-32 architecture. The Intel® Atom™ processor Z5xx series support IA-32 architecture.

The Intel® Xeon® processor E5 family, Intel® Xeon® processor E3 family, Intel® Core™ i7-3930K processor, 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series, Intel® Xeon® processor E7-8800/4800/2800 product families, Intel® Xeon® processor 3000, 3200, 5000, 5100, 5200, 5300, 5400, 7100, 7200, 7300, 7400 series, Intel® Core™2 Duo, Intel® Core™2 Extreme processors, Intel Core 2 Quad processors, Pentium® D processors, Pentium® Dual-Core processor, newer generations of Pentium 4 and Intel Xeon processor family support Intel® 64 architecture.

IA-32 architecture is the instruction set architecture and programming environment for Intel's 32-bit microprocessors.

Intel® 64 architecture is the instruction set architecture and programming environment which is the superset of Intel's 32-bit and 64-bit architectures. It is compatible with the IA-32 architecture.



...

2. Updates to Chapter 1, Volume 2A

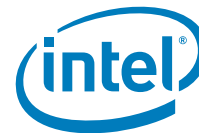
Change bars show changes to Chapter 1 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-L*.

...

1.1 INTEL® 64 AND IA-32 PROCESSORS COVERED IN THIS MANUAL

This manual set includes information pertaining primarily to the most recent Intel 64 and IA-32 processors, which include:

- Pentium® processors
- P6 family processors
- Pentium® 4 processors
- Pentium® M processors
- Intel® Xeon® processors
- Pentium® D processors
- Pentium® processor Extreme Editions
- 64-bit Intel® Xeon® processors
- Intel® Core™ Duo processor
- Intel® Core™ Solo processor
- Dual-Core Intel® Xeon® processor LV
- Intel® Core™2 Duo processor
- Intel® Core™2 Quad processor Q6000 series
- Intel® Xeon® processor 3000, 3200 series
- Intel® Xeon® processor 5000 series
- Intel® Xeon® processor 5100, 5300 series
- Intel® Core™2 Extreme processor X7000 and X6800 series
- Intel® Core™2 Extreme QX6000 series
- Intel® Xeon® processor 7100 series
- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® Xeon® processor 5200, 5400, 7400 series
- Intel® Core™2 Extreme processor QX9000 and X9000 series
- Intel® Core™2 Quad processor Q9000 series
- Intel® Core™2 Duo processor E8000, T9000 series
- Intel® Atom™ processor family
- Intel® Core™i7 processor



- Intel® Core™ i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Xeon® processor E5 family
- Intel® Xeon® processor E3 family
- Intel® Core™ i7-3930K processor
- 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series

P6 family processors are IA-32 processors based on the P6 family microarchitecture. This includes the Pentium® Pro, Pentium® II, Pentium® III, and Pentium® III Xeon® processors.

The Pentium® 4, Pentium® D, and Pentium® processor Extreme Editions are based on the Intel NetBurst® microarchitecture. Most early Intel® Xeon® processors are based on the Intel NetBurst® microarchitecture. Intel Xeon processor 5000, 7100 series are based on the Intel NetBurst® microarchitecture.

The Intel® Core™ Duo, Intel® Core™ Solo and dual-core Intel® Xeon® processor LV are based on an improved Pentium® M processor microarchitecture.

The Intel® Xeon® processor 3000, 3200, 5100, 5300, 7200, and 7300 series, Intel® Pentium® dual-core, Intel® Core™2 Duo, Intel® Core™2 Quad, and Intel® Core™2 Extreme processors are based on Intel® Core™ microarchitecture.

The Intel® Xeon® processor 5200, 5400, 7400 series, Intel® Core™2 Quad processor Q9000 series, and Intel® Core™2 Extreme processors QX9000, X9000 series, Intel® Core™2 processor E8000 series are based on Enhanced Intel® Core™ microarchitecture.

The Intel® Atom™ processor family is based on the Intel® Atom™ microarchitecture and supports Intel 64 architecture.

The Intel® Core™ i7 processor and the Intel® Core™ i5 processor are based on the Intel® microarchitecture code name Nehalem and support Intel 64 architecture.

Processors based on Intel® microarchitecture code name Westmere support Intel 64 architecture.

P6 family, Pentium® M, Intel® Core™ Solo, Intel® Core™ Duo processors, dual-core Intel® Xeon® processor LV, and early generations of Pentium 4 and Intel Xeon processors support IA-32 architecture. The Intel® Atom™ processor Z5xx series support IA-32 architecture.

The Intel® Xeon® processor E5 family, Intel® Xeon® processor E3 family, Intel® Core™ i7-3930K processor, 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series, Intel® Xeon® processor E7-8800/4800/2800 product families, Intel® Xeon® processor 3000, 3200, 5000, 5100, 5200, 5300, 5400, 7100, 7200, 7300, 7400 series, Intel® Core™2 Duo, Intel® Core™2 Extreme, Intel® Core™2 Quad processors, Pentium® D processors, Pentium® Dual-Core processor, newer generations of Pentium 4 and Intel Xeon processor family support Intel® 64 architecture.

IA-32 architecture is the instruction set architecture and programming environment for Intel's 32-bit microprocessors.

Intel® 64 architecture is the instruction set architecture and programming environment which is the superset of Intel's 32-bit and 64-bit architectures. It is compatible with the IA-32 architecture.



...

3. 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-L*.

...

AAS—ASCII Adjust AL After Subtraction

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
3F	AAS	NP	Invalid	Valid	ASCII adjust AL after subtraction.

Instruction Operand Encoding

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

Description

Adjusts the result of the subtraction of two unpacked BCD values to create a unpacked BCD result. The AL register is the implied source and destination operand for this instruction. The AAS instruction is only useful when it follows a SUB instruction that subtracts (binary subtraction) one unpacked BCD value from another and stores a byte result in the AL register. The AAA instruction then adjusts the contents of the AL register to contain the correct 1-digit unpacked BCD result.

If the subtraction produced a decimal carry, the AH register decrements by 1, and the CF and AF flags are set. If no decimal carry occurred, the CF and AF flags are cleared, and the AH register is unchanged. In either case, the AL register is left with its top four bits set to 0.

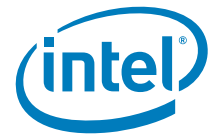
This instruction executes as described in compatibility mode and legacy mode. It is not valid in 64-bit mode.

Operation

```

IF 64-bit mode
  THEN
    #UD;
  ELSE
    IF ((AL AND 0FH) > 9) or (AF = 1)
      THEN
        AX ← AX - 6;
        AH ← AH - 1;
        AF ← 1;
        CF ← 1;
        AL ← AL AND 0FH;
      ELSE

```



CF ← 0;
 AF ← 0;
 AL ← AL AND 0FH;
 FI;
 FI;
 ...

CRC32 — Accumulate CRC32 Value

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
F2 0F 38 F0 /r	CRC32 r32, r/m8	RM	Valid	Valid	Accumulate CRC32 on r/m8.
F2 REX 0F 38 F0 /r	CRC32 r32, r/m8*	RM	Valid	N.E.	Accumulate CRC32 on r/m8.
F2 0F 38 F1 /r	CRC32 r32, r/m16	RM	Valid	Valid	Accumulate CRC32 on r/m16.
F2 0F 38 F1 /r	CRC32 r32, r/m32	RM	Valid	Valid	Accumulate CRC32 on r/m32.
F2 REX.W 0F 38 F0 /r	CRC32 r64, r/m8	RM	Valid	N.E.	Accumulate CRC32 on r/m8.
F2 REX.W 0F 38 F1 /r	CRC32 r64, r/m64	RM	Valid	N.E.	Accumulate CRC32 on r/m64.

NOTES:

*In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

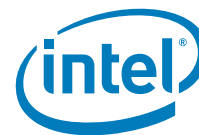
Instruction Operand Encoding

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

Description

Starting with an initial value in the first operand (destination operand), accumulates a CRC32 (polynomial 0x11EDC6F41) value for the second operand (source operand) and stores the result in the destination operand. The source operand can be a register or a memory location. The destination operand must be an r32 or r64 register. If the destination is an r64 register, then the 32-bit result is stored in the least significant double word and 00000000H is stored in the most significant double word of the r64 register.

The initial value supplied in the destination operand is a double word integer stored in the r32 register or the least significant double word of the r64 register. To incrementally accumulate a CRC32 value, software retains the result of the previous CRC32 operation in the destination operand, then executes the CRC32 instruction again with new input data in the source operand. Data contained in the source operand is processed in reflected bit order. This means that the most significant bit of the source operand is treated as the least significant bit of the quotient, and so on, for all the bits of the source operand. Likewise, the result of the CRC operation is stored in the destination operand in reflected bit order. This means that the most significant bit of the resulting CRC (bit 31) is stored in the least significant bit of the destination operand (bit 0), and so on, for all the bits of the CRC.



Operation

Notes:

BIT_REFLECT64: $DST[63-0] = SRC[0-63]$
BIT_REFLECT32: $DST[31-0] = SRC[0-31]$
BIT_REFLECT16: $DST[15-0] = SRC[0-15]$
BIT_REFLECT8: $DST[7-0] = SRC[0-7]$
MOD2: Remainder from Polynomial division modulus 2

CRC32 instruction for 64-bit source operand and 64-bit destination operand:

$TEMP1[63-0] \leftarrow BIT_REFLECT64(SRC[63-0])$
 $TEMP2[31-0] \leftarrow BIT_REFLECT32(DEST[31-0])$
 $TEMP3[95-0] \leftarrow TEMP1[63-0] \ll 32$
 $TEMP4[95-0] \leftarrow TEMP2[31-0] \ll 64$
 $TEMP5[95-0] \leftarrow TEMP3[95-0] \text{ XOR } TEMP4[95-0]$
 $TEMP6[31-0] \leftarrow TEMP5[95-0] \text{ MOD2 } 11EDC6F41H$
 $DEST[31-0] \leftarrow BIT_REFLECT(TEMP6[31-0])$
 $DEST[63-32] \leftarrow 00000000H$

CRC32 instruction for 32-bit source operand and 32-bit destination operand:

$TEMP1[31-0] \leftarrow BIT_REFLECT32(SRC[31-0])$
 $TEMP2[31-0] \leftarrow BIT_REFLECT32(DEST[31-0])$
 $TEMP3[63-0] \leftarrow TEMP1[31-0] \ll 32$
 $TEMP4[63-0] \leftarrow TEMP2[31-0] \ll 32$
 $TEMP5[63-0] \leftarrow TEMP3[63-0] \text{ XOR } TEMP4[63-0]$
 $TEMP6[31-0] \leftarrow TEMP5[63-0] \text{ MOD2 } 11EDC6F41H$
 $DEST[31-0] \leftarrow BIT_REFLECT(TEMP6[31-0])$

CRC32 instruction for 16-bit source operand and 32-bit destination operand:

$TEMP1[15-0] \leftarrow BIT_REFLECT16(SRC[15-0])$
 $TEMP2[31-0] \leftarrow BIT_REFLECT32(DEST[31-0])$
 $TEMP3[47-0] \leftarrow TEMP1[15-0] \ll 32$
 $TEMP4[47-0] \leftarrow TEMP2[31-0] \ll 16$
 $TEMP5[47-0] \leftarrow TEMP3[47-0] \text{ XOR } TEMP4[47-0]$
 $TEMP6[31-0] \leftarrow TEMP5[47-0] \text{ MOD2 } 11EDC6F41H$
 $DEST[31-0] \leftarrow BIT_REFLECT(TEMP6[31-0])$

CRC32 instruction for 8-bit source operand and 64-bit destination operand:

$TEMP1[7-0] \leftarrow BIT_REFLECT8(SRC[7-0])$
 $TEMP2[31-0] \leftarrow BIT_REFLECT32(DEST[31-0])$
 $TEMP3[39-0] \leftarrow TEMP1[7-0] \ll 32$
 $TEMP4[39-0] \leftarrow TEMP2[31-0] \ll 8$
 $TEMP5[39-0] \leftarrow TEMP3[39-0] \text{ XOR } TEMP4[39-0]$
 $TEMP6[31-0] \leftarrow TEMP5[39-0] \text{ MOD2 } 11EDC6F41H$
 $DEST[31-0] \leftarrow BIT_REFLECT(TEMP6[31-0])$
 $DEST[63-32] \leftarrow 00000000H$

CRC32 instruction for 8-bit source operand and 32-bit destination operand:

$TEMP1[7-0] \leftarrow BIT_REFLECT8(SRC[7-0])$



```
TEMP2[31-0] ← BIT_REFLECT32 (DEST[31-0])
TEMP3[39-0] ← TEMP1[7-0] « 32
TEMP4[39-0] ← TEMP2[31-0] « 8
TEMP5[39-0] ← TEMP3[39-0] XOR TEMP4[39-0]
TEMP6[31-0] ← TEMP5[39-0] MOD2 11EDC6F41H
DEST[31-0] ← BIT_REFLECT (TEMP6[31-0])
```

Flags Affected

None

Intel C/C++ Compiler Intrinsic Equivalent

```
unsigned int __mm_crc32_u8( unsigned int crc, unsigned char data )
unsigned int __mm_crc32_u16( unsigned int crc, unsigned short data )
unsigned int __mm_crc32_u32( unsigned int crc, unsigned int data )
unsigned __int64 __mm_crc32_u64( unsigned __int64 crc, unsigned __int64 data )
```

SIMD Floating Point Exceptions

None

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segments.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0. If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0. If LOCK prefix is used.

Virtual 8086 Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.
#UD	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.



If LOCK prefix is used.

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.
- #AC(0) If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
- #UD If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.
If LOCK prefix is used.

...

LAR—Load Access Rights Byte

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
0F 02 /r	LAR <i>r16, r16/m16</i>	RM	Valid	Valid	<i>r16</i> ← access rights referenced by <i>r16/m16</i>
0F 02 /r	LAR <i>reg, r32/m16</i> ¹	RM	Valid	Valid	<i>reg</i> ← access rights referenced by <i>r32/m16</i>

NOTES:

1. For all loads (regardless of source or destination sizing) only bits 16-0 are used. Other bits are ignored.

Instruction Operand Encoding

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

Description

Loads the access rights from the segment descriptor specified by the second operand (source operand) into the first operand (destination operand) and sets the ZF flag in the flag register. The source operand (which can be a register or a memory location) contains the segment selector for the segment descriptor being accessed. If the source operand is a memory address, only 16 bits of data are accessed. The destination operand is a general-purpose register.

The processor performs access checks as part of the loading process. Once loaded in the destination register, software can perform additional checks on the access rights information.

The access rights for a segment descriptor include fields located in the second doubleword (bytes 4–7) of the segment descriptor. The following fields are loaded by the LAR instruction:

- Bits 7:0 are returned as 0



- Bits 11:8 return the segment type.
- Bit 12 returns the S flag.
- Bits 14:13 return the DPL.
- Bit 15 returns the P flag.
- The following fields are returned only if the operand size is greater than 16 bits:
 - Bits 19:16 are undefined.
 - Bit 20 returns the software-available bit in the descriptor.
 - Bit 21 returns the L flag.
 - Bit 22 returns the D/B flag.
 - Bit 23 returns the G flag.
 - Bits 31:24 are returned as 0.

This instruction performs the following checks before it loads the access rights in the destination register:

- Checks that the segment selector is not NULL.
- Checks that the segment selector points to a descriptor that is within the limits of the GDT or LDT being accessed
- Checks that the descriptor type is valid for this instruction. All code and data segment descriptors are valid for (can be accessed with) the LAR instruction. The valid system segment and gate descriptor types are given in Table 3-62.
- If the segment is not a conforming code segment, it checks that the specified segment descriptor is visible at the CPL (that is, if the CPL and the RPL of the segment selector are less than or equal to the DPL of the segment selector).

If the segment descriptor cannot be accessed or is an invalid type for the instruction, the ZF flag is cleared and no access rights are loaded in the destination operand.

The LAR instruction can only be executed in protected mode and IA-32e mode.

Table 3-62 Segment and Gate Types

Type	Protected Mode		IA-32e Mode	
	Name	Valid	Name	Valid
0	Reserved	No	Reserved	No
1	Available 16-bit TSS	Yes	Reserved	No
2	LDT	Yes	LDT	No
3	Busy 16-bit TSS	Yes	Reserved	No
4	16-bit call gate	Yes	Reserved	No
5	16-bit/32-bit task gate	Yes	Reserved	No
6	16-bit interrupt gate	No	Reserved	No
7	16-bit trap gate	No	Reserved	No
8	Reserved	No	Reserved	No
9	Available 32-bit TSS	Yes	Available 64-bit TSS	Yes
A	Reserved	No	Reserved	No
B	Busy 32-bit TSS	Yes	Busy 64-bit TSS	Yes



Table 3-62 Segment and Gate Types

C	32-bit call gate	Yes	64-bit call gate	Yes
D	Reserved	No	Reserved	No
E	32-bit interrupt gate	No	64-bit interrupt gate	No
F	32-bit trap gate	No	64-bit trap gate	No

Operation

```

IF Offset(SRC) > descriptor table limit
  THEN
    ZF ← 0;
  ELSE
    SegmentDescriptor ← descriptor referenced by SRC;
    IF SegmentDescriptor(Type) ≠ conforming code segment
      and (CPL > DPL) or (RPL > DPL)
      or SegmentDescriptor(Type) is not valid for instruction
      THEN
        ZF ← 0;
      ELSE
        DEST ← access rights from SegmentDescriptor as given in Description section;
        ZF ← 1;
    FI;
  FI;

```

Flags Affected

The ZF flag is set to 1 if the access rights are loaded successfully; otherwise, it is cleared to 0.

Protected Mode Exceptions

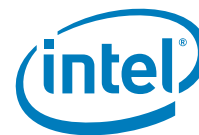
- #GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
If the DS, ES, FS, or GS register is used to access memory and it contains a NULL segment selector.
- #SS(0) If a memory operand effective address is outside the SS segment limit.
- #PF(fault-code) If a page fault occurs.
- #AC(0) If alignment checking is enabled and the memory operand effective address is unaligned while the current privilege level is 3.
- #UD If the LOCK prefix is used.

Real-Address Mode Exceptions

- #UD The LAR instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

- #UD The LAR instruction cannot be executed in virtual-8086 mode.



Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If the memory operand effective address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory operand effective address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and the memory operand effective address is unaligned while the current privilege level is 3.
#UD	If the LOCK prefix is used.
...	

LOCK—Assert LOCK# Signal Prefix

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
F0	LOCK	NP	Valid	Valid	Asserts LOCK# signal for duration of the accompanying instruction.

NOTES:

* See IA-32 Architecture Compatibility section below.

Instruction Operand Encoding

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

Description

Causes the processor's LOCK# signal to be asserted during execution of the accompanying instruction (turns the instruction into an atomic instruction). In a multiprocessor environment, the LOCK# signal ensures that the processor has exclusive use of any shared memory while the signal is asserted.

Note that, in later Intel 64 and IA-32 processors (including the Pentium 4, Intel Xeon, and P6 family processors), locking may occur without the LOCK# signal being asserted. See the "IA-32 Architecture Compatibility" section below.

The LOCK prefix can be prepended only to the following instructions and only to those forms of the instructions where the destination operand is a memory operand: ADD, ADC, AND, BTC, BTR, BTS, CMPXCHG, CMPXCH8B, CMPXCHG16B, DEC, INC, NEG, NOT, OR, SBB, SUB, XOR, XADD, and XCHG. If the LOCK prefix is used with one of these instructions and the source operand is a memory operand, an undefined opcode exception (#UD) may be generated. An undefined opcode exception will also be generated if the LOCK prefix is used with any instruction not in the above list. The XCHG instruction always asserts the LOCK# signal regardless of the presence or absence of the LOCK prefix.

The LOCK prefix is typically used with the BTS instruction to perform a read-modify-write operation on a memory location in shared memory environment.



The integrity of the LOCK prefix is not affected by the alignment of the memory field. Memory locking is observed for arbitrarily misaligned fields.

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

...

4. Updates to Chapter 4, Volume 2B

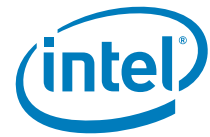
Change bars show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, M-Z*.

...

MASKMOVDQU—Store Selected Bytes of Double Quadword

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F F7 /r MASKMOVDQU <i>xmm1</i> , <i>xmm2</i>	RM	V/V	SSE2	Selectively write bytes from <i>xmm1</i> to memory location using the byte mask in <i>xmm2</i> . The default memory location is specified by DS:DI/EDI/RDI.
VEX.128.66.0F.WIG F7 /r VMASKMOVDQU <i>xmm1</i> , <i>xmm2</i>	RM	V/V	AVX	Selectively write bytes from <i>xmm1</i> to memory location using the byte mask in <i>xmm2</i> . The default memory location is specified by DS:DI/EDI/RDI.

...



MFENCE—Memory Fence

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF AE /6	MFENCE	NP	Valid	Valid	Serializes load and store operations.

Instruction Operand Encoding

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

Description

Performs a serializing operation on all load-from-memory and store-to-memory instructions that were issued prior the MFENCE instruction. This serializing operation guarantees that every load and store instruction that precedes the MFENCE instruction in program order becomes globally visible before any load or store instruction that follows the MFENCE instruction.¹ The MFENCE instruction is ordered with respect to all load and store instructions, other MFENCE instructions, any LFENCE and SFENCE instructions, and any serializing instructions (such as the CPUID instruction). MFENCE does not serialize the instruction stream.

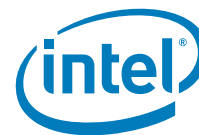
Weakly ordered memory types can be used to achieve higher processor performance through such techniques as out-of-order issue, speculative reads, write-combining, and write-collapsing. The degree to which a consumer of data recognizes or knows that the data is weakly ordered varies among applications and may be unknown to the producer of this data. The MFENCE instruction provides a performance-efficient way of ensuring load and store ordering between routines that produce weakly-ordered results and routines that consume that data.

Processors are free to fetch and cache data speculatively from regions of system memory that use the WB, WC, and WT memory types. This speculative fetching can occur at any time and is not tied to instruction execution. Thus, it is not ordered with respect to executions of the MFENCE instruction; data can be brought into the caches speculatively just before, during, or after the execution of an MFENCE instruction.

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

...

1. A load instruction is considered to become globally visible when the value to be loaded into its destination register is determined.



MONITOR—Set Up Monitor Address

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
0F 01 C8	MONITOR	NP	Valid	Valid	Sets up a linear address range to be monitored by hardware and activates the monitor. The address range should be a write-back memory caching type. The address is DS:EAX (DS:RAX in 64-bit mode).

Instruction Operand Encoding

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

Description

The MONITOR instruction arms address monitoring hardware using an address specified in EAX (the address range that the monitoring hardware checks for store operations can be determined by using CPUID). A store to an address within the specified address range triggers the monitoring hardware. The state of monitor hardware is used by MWAIT.

The content of EAX is an effective address (in 64-bit mode, RAX is used). By default, the DS segment is used to create a linear address that is monitored. Segment overrides can be used.

ECX and EDX are also used. They communicate other information to MONITOR. ECX specifies optional extensions. EDX specifies optional hints; it does not change the architectural behavior of the instruction. For the Pentium 4 processor (family 15, model 3), no extensions or hints are defined. Undefined hints in EDX are ignored by the processor; undefined extensions in ECX raises a general protection fault.

The address range must use memory of the write-back type. Only write-back memory will correctly trigger the monitoring hardware. Additional information on determining what address range to use in order to prevent false wake-ups is described in Chapter 8, "Multiple-Processor Management" of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

The MONITOR instruction is ordered as a load operation with respect to other memory transactions. The instruction is subject to the permission checking and faults associated with a byte load. Like a load, MONITOR sets the A-bit but not the D-bit in page tables.

CPUID.01H:ECX.MONITOR[bit 3] indicates the availability of MONITOR and MWAIT in the processor. When set, MONITOR 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_ENABLE MSR; disabling MONITOR clears the CPUID feature flag and causes execution to generate an invalid-opcode exception.

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

...



MOVHLPS— Move Packed Single-Precision Floating-Point Values High to Low

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
OF 12 /r MOVHLPS <i>xmm1</i> , <i>xmm2</i>	RM	V/V	SSE	Move two packed single-precision floating-point values from high quadword of <i>xmm2</i> to low quadword of <i>xmm1</i> .
VEX.NDS.128.OF.WIG 12 /r VMOVHLPS <i>xmm1</i> , <i>xmm2</i> , <i>xmm3</i>	RVM	V/V	AVX	Merge two packed single-precision floating-point values from high quadword of <i>xmm3</i> and low quadword of <i>xmm2</i> .

...

MWAIT—Monitor Wait

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
OF 01 C9	MWAIT	NP	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
NP	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.

CPUID.01H:ECX.MONITOR[bit 3] 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_ENABLE MSR; disabling MWAIT clears the CPUID feature flag and causes execution to generate an invalid-opcode exception.

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

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



implement MWAIT supported only the zero value for EAX and ECX. Later processors allowed setting ECX[0] to enable masked interrupts as break events for MWAIT (see below). Software can use the CPUID instruction to determine the extensions and hints supported by the processor.

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.

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 either (1) if the interrupt would be delivered to software (e.g., as it would be if HLT had been executed instead of MWAIT); or (2) if ECX[0] = 1. Software can execute MWAIT with ECX[0] = 1 only if CPUID.05H:ECX[bit 1] = 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[bit 0] reporting 1.

EAX and ECX are 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 4-10 describes the meaning of ECX and EAX registers for MWAIT extensions.



Table 4-10 MWAIT Extension Register (ECX)

Bits	Description
0	Treat interrupts as break events even if masked (e.g., even if EFLAGS.IF=0). May be set only if CPUID.05H:ECX[bit 1] = 1.
31: 1	Reserved

Table 4-11 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;
}
```

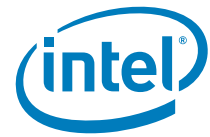
Intel C/C++ Compiler Intrinsic Equivalent

```
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 1] = 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 1] = 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).
-----	--

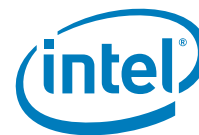
Compatibility Mode Exceptions

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 1] = 0.
#UD	If the current privilege level is not 0. If CPUID.01H:ECX.MONITOR[bit 3] = 0.

...



PAUSE—Spin Loop Hint

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
F3 90	PAUSE	NP	Valid	Valid	Gives hint to processor that improves performance of spin-wait loops.

Instruction Operand Encoding

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

Description

Improves the performance of spin-wait loops. When executing a “spin-wait loop,” processors will suffer a severe performance penalty when exiting the loop because it detects a possible memory order violation. The PAUSE instruction provides a hint to the processor that the code sequence is a spin-wait loop. The processor uses this hint to avoid the memory order violation in most situations, which greatly improves processor performance. For this reason, it is recommended that a PAUSE instruction be placed in all spin-wait loops.

An additional function of the PAUSE instruction is to reduce the power consumed by a processor while executing a spin loop. A processor can execute a spin-wait loop extremely quickly, causing the processor to consume a lot of power while it waits for the resource it is spinning on to become available. Inserting a pause instruction in a spin-wait loop greatly reduces the processor’s power consumption.

This instruction was introduced in the Pentium 4 processors, but is backward compatible with all IA-32 processors. In earlier IA-32 processors, the PAUSE instruction operates like a NOP instruction. The Pentium 4 and Intel Xeon processors implement the PAUSE instruction as a delay. The delay is finite and can be zero for some processors. This instruction does not change the architectural state of the processor (that is, it performs essentially a delaying no-op operation).

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

...

POPCNT – Return the Count of Number of Bits Set to 1

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
F3 0F B8 /r	POPCNT <i>r16, r/m16</i>	RM	Valid	Valid	POPCNT on <i>r/m16</i>
F3 0F B8 /r	POPCNT <i>r32, r/m32</i>	RM	Valid	Valid	POPCNT on <i>r/m32</i>
F3 REX.W 0F B8 /r	POPCNT <i>r64, r/m64</i>	RM	Valid	N.E.	POPCNT on <i>r/m64</i>



Instruction Operand Encoding

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

Description

This instruction calculates of number of bits set to 1 in the second operand (source) and returns the count in the first operand (a destination register).

Operation

```
Count = 0;
For (i=0; i < OperandSize; i++)
{
    IF (SRC[ i] = 1) // i'th bit
        THEN Count++; FI;
}
DEST ← Count;
```

Flags Affected

OF, SF, ZF, AF, CF, PF are all cleared. ZF is set if SRC = 0, otherwise ZF is cleared

Intel C/C++ Compiler Intrinsic Equivalent

```
POPCNT:    int _mm_popcnt_u32(unsigned int a);
```

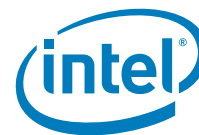
```
POPCNT:    int64_t _mm_popcnt_u64(unsigned __int64 a);
```

Protected Mode Exceptions

- #GP(0) If a memory operand effective address is outside the CS, DS, ES, FS or GS segments.
- #SS(0) If a memory operand effective address is outside the SS segment limit.
- #PF (fault-code) For a page fault.
- #AC(0) If an unaligned memory reference is made while the current privilege level is 3 and alignment checking is enabled.
- #UD If CPUID.01H:ECX.POPCNT [Bit 23] = 0.
If LOCK prefix is used.
Either the prefix REP (F3h) or REPN (F2H) is used.

Real Mode Exceptions

- #GP(0) If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
- #SS(0) If a memory operand effective address is outside the SS segment limit.
- #UD If CPUID.01H:ECX.POPCNT [Bit 23] = 0.
If LOCK prefix is used.
Either the prefix REP (F3h) or REPN (F2H) is used.



Virtual 8086 Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#AC(0)	If an unaligned memory reference is made while alignment checking is enabled.
#UD	If CPUID.01H:ECX.POPCNT [Bit 23] = 0. If LOCK prefix is used. Either the prefix REP (F3h) or REPN (F2H) is used.

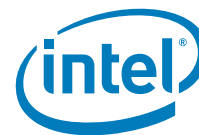
Compatibility Mode Exceptions

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.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If CPUID.01H:ECX.POPCNT [Bit 23] = 0. If LOCK prefix is used. Either the prefix REP (F3h) or REPN (F2H) is used.

...



WRMSR—Write to Model Specific Register

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 30	WRMSR	NP	Valid	Valid	Write the value in EDX:EAX to MSR specified by ECX.

Instruction Operand Encoding

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

Description

Writes the contents of registers EDX:EAX into the 64-bit model specific register (MSR) specified in the ECX register. (On processors that support the Intel 64 architecture, the high-order 32 bits of RCX are ignored.) The contents of the EDX register are copied to high-order 32 bits of the selected MSR and the contents of the EAX register are copied to low-order 32 bits of the MSR. (On processors that support the Intel 64 architecture, the high-order 32 bits of each of RAX and RDX are ignored.) Undefined or reserved bits in an MSR should be set to values previously read.

This instruction must be executed at privilege level 0 or in real-address mode; otherwise, a general protection exception #GP(0) is generated. Specifying a reserved or unimplemented MSR address in ECX will also cause a general protection exception. The processor will also generate a general protection exception if software attempts to write to bits in a reserved MSR.

When the WRMSR instruction is used to write to an MTRR, the TLBs are invalidated. This includes global entries (see “Translation Lookaside Buffers (TLBs)” in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*).

MSRs control functions for testability, execution tracing, performance-monitoring and machine check errors. Chapter 34, “Model-Specific Registers (MSRs)”, in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C*, lists all MSRs that can be written with this instruction and their addresses. Note that each processor family has its own set of MSRs.

The WRMSR instruction is a serializing instruction (see “Serializing Instructions” in Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*). Note that WRMSR to the IA32_TSC_DEADLINE MSR (MSR index 6E0H) and the X2APIC MSRs (MSR indices 802H to 83FH) are not serializing.

The CPUID instruction should be used to determine whether MSRs are supported (CPUID.01H:EDX[5] = 1) before using this instruction.

...

5. Updates to Appendix A, Volume 2C

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

...



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

	0	1	2	3	4	5	6	7	
0	ADD Eb, Gb Ev, Gv Gb, Eb Gv, Ev AL, lb rAX, lz						PUSH ES ⁶⁴	POP ES ⁶⁴	
1	ADC Eb, Gb Ev, Gv Gb, Eb Gv, Ev AL, lb rAX, lz						PUSH SS ⁶⁴	POP SS ⁶⁴	
2	AND Eb, Gb Ev, Gv Gb, Eb Gv, Ev AL, lb rAX, lz						SEG=ES (Prefix)	DAA ⁶⁴	
3	XOR Eb, Gb Ev, Gv Gb, Eb Gv, Ev AL, lb rAX, lz						SEG=SS (Prefix)	AAA ⁶⁴	
4	INC ⁶⁴ general register / REX ⁰⁶⁴ 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 ⁶⁴ / PUSHAD ⁶⁴	POPA ⁶⁴ / POPAD ⁶⁴	BOUND ⁶⁴ Gv, Ma	ARPL ⁶⁴ Ew, Gw MOVXD ⁰⁶⁴ Gv, Ev	SEG=FS (Prefix)	SEG=GS (Prefix)	Operand Size (Prefix)	Address Size (Prefix)	
7	Jcc ⁶⁴ , 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} Eb, lb Ev, lz Eb, lb ⁶⁴ Ev, lb				TEST Eb, Gb Ev, Gv		XCHG 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 AL, Ob rAX, Ov Ob, AL Ov, rAX				MOVS/B Yb, Xb	MOVS/W/D/Q Yv, Xv	CMPS/B Xb, Yb	CMPS/W/D Xv, Yv	
B	MOV immediate byte into byte register AL/R8L, lb CL/R9L, lb DL/R10L, lb BL/R11L, lb AH/R12L, lb CH/R13L, lb DH/R14L, lb BH/R15L, lb								
C	Shift Grp 2 ^{1A} Eb, lb Ev, lb		RETN ⁶⁴ lw	RETN ⁶⁴	LES ⁶⁴ Gz, Mp VEX+2byte	LDS ⁶⁴ Gz, Mp VEX+1byte	Grp 11 ^{1A} - MOV Eb, lb Ev, lz		
D	Shift Grp 2 ^{1A} Eb, 1 Ev, 1 Eb, CL Ev, CL				AAM ⁶⁴ lb	AAD ⁶⁴ lb		XLAT/ XLATB	
E	LOOPNE ⁶⁴ / LOOPNZ ⁶⁴ Jb	LOOPE ⁶⁴ / LOOPZ ⁶⁴ Jb	LOOP ⁶⁴ Jb	JrCXZ ⁶⁴ / Jb	IN AL, lb eAX, lb		OUT lb, AL lb, eAX		
F	LOCK (Prefix)		REPNE XACQUIRE (Prefix)	REP/REPE XRELEASE (Prefix)	HLT	CMC	Unary Grp 3 ^{1A} Eb Ev		

...

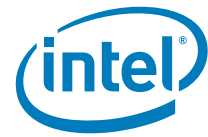


Table A-3 Two-byte Opcode Map: 00H – 77H (First Byte is 0FH) *

pxf	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		vmovups	vmovlps Vq, Hq, Mq vmovhlps Vq, Hq, Uq	vmovlps Mq, Vq	vunpcklps Vx, Hx, Wx	vunpckhps Vx, Hx, Wx	vmovhps ^{V1} Vdq, Hq, Mq vmovhlps Vdq, Hq, Uq	vmovhps ^{V1} Mq, Vq	
	66	vmovupd	vmovlpd Vq, Hq, Mq	vmovlpd Mq, Vq	vunpcklpd Vx,Hx,Wx	vunpckhpd Vx,Hx,Wx	vmovhpd ^{V1} Vdq, Hq, Mq	vmovhpd ^{V1} Mq, Vq	
	F3	vmovss Vx, Hx, Wss	vmovss Wss, Hx, Vss	vmovsldup Vx, Wx			vmovshdup Vx, Wx		
	F2	vmovsd Vx, Hx, Wsd	vmovsd Wsd, Hx, Vsd	vmovddup Vx, Wx					
2	MOV Rd, Cd	MOV Rd, Dd	MOV Cd, Rd	MOV Dd, Rd					
3	WRMSR	RDTSC	RDMSR	RDPIC	SYSENTER	SYSEXIT		GETSEC	
4	CMOVcc, (Gv, Ev) - Conditional Move								
	O	NO	B/C/NAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE	
5		vmovmskps Gy, Ups	vsqrtps Vps, Wps	vrsqrtps Vps, Wps	vrcpps Vps, Wps	vandps Vps, Hps, Wps	vandnps Vps, Hps, Wps	vorps Vps, Hps, Wps	vxorps Vps, Hps, Wps
	66	vmovmskpd Gy, Upd	vsqrtpd Vpd, Wpd			vandpd Vpd, Hpd, Wpd	vandnpd Vpd, Hpd, Wpd	vorpd Vpd, Hpd, Wpd	vxorpd Vpd, Hpd, Wpd
	F3		vsqrtss Vss, Hss, Wss	vrsqrtss Vss, Hss, Wss	vrcpss Vss, Hss, Wss				
	F2		vsqrtsd Vsd, Hsd, 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	vpunpcklbw Vx, Hx, Wx	vpunpcklwd Vx, Hx, Wx	vpunpckldq Vx, Hx, Wx	vpacksswb Vx, Hx, Wx	vpcmpgtb Vx, Hx, Wx	vpcmpgtw Vx, Hx, Wx	vpcmpgtd Vx, Hx, Wx	vpackuswb Vx, Hx, Wx
	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 vzeroupper ^V vzeroall ^V
	66	vpshufd Vx, Wx, Ib				vpcmpeqb Vx, Hx, Wx	vpcmpeqw Vx, Hx, Wx	vpcmpeqd Vx, Hx, Wx	
	F3	vpshufw Vx, Wx, Ib							
	F2	vpshufw Vx, Wx, Ib							



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

	pxf	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		vmovaps Vps, Wps	vmovaps Wps, Vps	cvtpi2ps Vps, Qpi	vmovntps Mps, Vps	cvtps2pi Ppi, Wps	cvtps2pi Ppi, Wps	vucomiss Vss, Wss	vcomiss Vss, Wss
	66	vmovapd Vpd, Wpd	vmovapd Wpd, Vpd	cvtpi2pd Vpd, Qpi	vmovntpd Mpd, Vpd	cvtpd2pi Ppi, Wpd	cvtpd2pi Qpi, Wpd	vucomisd Vsd, Wsd	vcomisd Vsd, Wsd
	F3			vcvtsi2ss Vss, Hss, Ey		vcvtss2si Gy, Wss	vcvtss2si Gy, Wss		
	F2			vcvtsi2sd Vsd, Hsd, Ey		vcvtss2si Gy, Wsd	vcvtss2si Gy, Wsd		
3		3-byte escape (Table A-4)		3-byte escape (Table A-5)					
4		CMOVcc(Gv, Ev) - Conditional Move							
		S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
5		vaddps Vps, Hps, Wps	vmulps Vps, Hps, Wps	vcvtps2pd Vpd, Wps	vcvtdq2ps Vps, Wdq	vsubps Vps, Hps, Wps	vminps Vps, Hps, Wps	vdivps Vps, Hps, Wps	vmaxps Vps, Hps, Wps
	66	vaddpd Vpd, Hpd, Wpd	vmulpd Vpd, Hpd, Wpd	vcvtpd2ps Vps, Wpd	vcvtps2dq Vdq, Wps	vsubpd Vpd, Hpd, Wpd	vminpd Vpd, Hpd, Wpd	vdivpd Vpd, Hpd, Wpd	vmaxpd Vpd, Hpd, Wpd
	F3	vaddss Vss, Hss, Wss	vmulss Vss, Hss, Wss	vcvtss2sd Vsd, Hs, Wss	vcvtps2dq Vdq, Wps	vsubss Vss, Hss, Wss	vminss Vss, Hss, Wss	vdivss Vss, Hss, Wss	vmaxss Vss, Hss, Wss
	F2	vaddsd Vsd, Hsd, Wsd	vmulsd Vsd, Hsd, Wsd	vcvtss2sd Vss, Hs, Wsd		vsubsd Vsd, Hsd, Wsd	vminsd Vsd, Hsd, Wsd	vdivsd Vsd, Hsd, Wsd	vmaxsd Vsd, Hsd, Wsd
6		punpckhbw Pq, Qd	punpckhwd Pq, Qd	punpckhdq Pq, Qd	packssdw Pq, Qd			movd/q Pd, Ey	movq Pq, Qq
	66	vpunpckhbw Vx, Hx, Wx	vpunpckhwd Vx, Hx, Wx	vpunpckhdq Vx, Hx, Wx	vpackssdw Vx, Hx, Wx	vpunpckldq Vx, Hx, Wx	vpunpckhdq Vx, Hx, Wx	vmovd/q Vy, Ey	vmovdqa Vx, Wx
	F3								vmovdqu Vx, Wx
7		VMREAD Ey, Gy	VMWRITE Gy, Ey					movd/q Ey, Pd	movq Qq, Pq
	66					vhaddpd Vpd, Hpd, Wpd	vhsbpd Vpd, Hpd, Wpd	vmovd/q Ey, Vy	vmovdqa Wx, Vx
	F3							vmovq Vq, Wq	vmovdqu Wx, Vx
	F2					vhaddps Vps, Hps, Wps	vhsbps Vps, Hps, Wps		

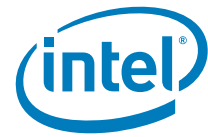


Table A-3. Two-byte Opcode Map: 80H – F7H (First Byte is 0FH) *

px	0	1	2	3	4	5	6	7	
8	Jcc ⁶⁴ , 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/CNAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE	
A	PUSH ⁶⁴ FS	POP ⁶⁴ FS	CPUID	BT Ev, Gv	SHLD Ev, Gv, Ib	SHLD Ev, Gv, CL			
B	CMPXCHG Eb, Gb		LSS Gv, Mp	BTR Ev, Gv	LFS Gv, Mp	LGS Gv, Mp	MOVZX Gv, Eb		
								Gv, Ew	
C		XADD Eb, Gb	XADD Ev, Gv	vcmpps Vps,Hps,Wps,Ib	movnti My, Gy	pinsrw Pq,Ry/Mw,Ib	pextrw Gd, Nq, Ib	vshufps Vps,Hps,Wps,Ib	Grp 9 ^{1A}
	66			vcmppd Vpd,Hpd,Wpd,Ib		vpinsrw Vdq,Hdq,Ry/Mw,Ib	vpextrw Gd, Udq, Ib	vshufpd Vpd,Hpd,Wpd,Ib	
	F3			vcmpss Vss,Hss,Wss,Ib					
	F2			vcmpsd Vsd,Hsd,Wsd,Ib					
D			psrlw Pq, Qq	psrld Pq, Qq	psrlq Pq, Qq	paddq Pq, Qq	pmullw Pq, Qq		pmovmskb Gd, Nq
	66	vaddsubpd Vpd, Hpd, Wpd	vpsrlw Vx, Hx, Wx	vpsrld Vx, Hx, Wx	vpsrlq Vx, Hx, Wx	vpaddq Vx, Hx, Wx	vpmullw Vx, Hx, Wx	vmovq Wq, Vq	vpmovmskb Gd, Ux
	F3							movq2dq Vdq, Nq	
	F2	vaddsubps Vps, Hps, Wps						movdq2q Pq, Uq	
E		pavgb Pq, Qq	psraw Pq, Qq	psrad Pq, Qq	pavgw Pq, Qq	pmulhw Pq, Qq	pmulhw Pq, Qq		movntq Mq, Pq
	66	vpavgb Vx, Hx, Wx	vpsraw Vx, Hx, Wx	vpsrad Vx, Hx, Wx	vpavgw Vx, Hx, Wx	vpmulhw Vx, Hx, Wx	vpmulhw Vx, Hx, Wx	vcvttd2dq Vx, Wpd	vmovntdq Mx, Vx
	F3							vcvtdd2pd Vx, Wpd	
	F2							vcvtdd2dq Vx, Wpd	
F			psllw Pq, Qq	pslld Pq, Qq	psllq Pq, Qq	pmuludq Pq, Qq	pmaddwd Pq, Qq	psadbw Pq, Qq	maskmovq Pq, Nq
	66		vpsllw Vx, Hx, Wx	vpslld Vx, Hx, Wx	vpsllq Vx, Hx, Wx	vpmuludq Vx, Hx, Wx	vpmaddwd Vx, Hx, Wx	vpsadbw Vx, Hx, Wx	vmaskmovdqu Vdq, Udq
	F2	vlddqu Vx, Mx							

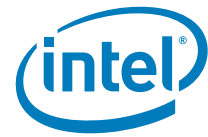


Table A-3. Two-byte Opcode Map: 88H – FFH (First Byte is 0FH) *

px	8	9	A	B	C	D	E	F	
8	Jcc ⁶⁴ , 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 ⁶⁴ GS	POP ⁶⁴ GS	RSM	BTS Ev, Gv	SHRD Ev, Gv, Ib	SHRD Ev, Gv, CL	(Grp 15 ^{1A}) ^{1C}	IMUL Gv, Ev	
B	<i>JMPE</i> (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	MOVSBX Gv, Eb Gv, Ew		
	F3			POPCNT Gv, Ev		TZCNT Gv, Ev	LZCNT 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	vpsubusb Vx, Hx, Wx	vpsubusw Vx, Hx, Wx	vpminub Vx, Hx, Wx	vpand Vx, Hx, Wx	vpaddusb Vx, Hx, Wx	vpaddusw Vx, Hx, Wx	vpmaxub Vx, Hx, Wx	vpandn Vx, Hx, Wx
	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	vpsubsb Vx, Hx, Wx	vpsubsw Vx, Hx, Wx	vpminsw Vx, Hx, Wx	vpior Vx, Hx, Wx	vpaddsb Vx, Hx, Wx	vpaddsw Vx, Hx, Wx	vpmaxsw Vx, Hx, Wx	vpior Vx, Hx, Wx
	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	vpsubb Vx, Hx, Wx	vpsubw Vx, Hx, Wx	vpsubd Vx, Hx, Wx	vpsubq Vx, Hx, Wx	vpaddb Vx, Hx, Wx	vpaddw Vx, Hx, Wx	vpaddd Vx, Hx, Wx	
	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.

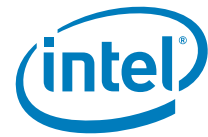


Table A-4 Three-byte Opcode Map: 00H — F7H (First Two Bytes are 0F 38H) *

px	0	1	2	3	4	5	6	7
0	pshufb Pq, Qq	phaddw Pq, Qq	phadd Pq, Qq	phaddsw Pq, Qq	pmaddubsw Pq, Qq	phsubw Pq, Qq	phsubd Pq, Qq	phsubsw Pq, Qq
66	vpshufb Vx, Hx, Wx	vphaddw Vx, Hx, Wx	vphadd Vx, Hx, Wx	vphaddsw Vx, Hx, Wx	vpaddubsw Vx, Hx, Wx	vphsubw Vx, Hx, Wx	vphsubd Vx, Hx, Wx	vphsubsw Vx, Hx, Wx
1	pblendvb Vdq, Wdq			vcvtph2ps ^Y Vx, Wx, Ib	blendvps Vdq, Wdq	blendvpd Vdq, Wdq	vpermps ^Y Vqq, Hqq, Wqq	vptest Vx, Wx
2	vpmovsxbw Vx, Ux/Mq	vpmovsxbd Vx, Ux/Md	vpmovsxbq Vx, Ux/Mw	vpmovsxwd Vx, Ux/Mq	vpmovsxwq Vx, Ux/Md	vpmovsxdq Vx, Ux/Mq		
3	vpmovzxbw Vx, Ux/Mq	vpmovzxbd Vx, Ux/Md	vpmovzxbq Vx, Ux/Mw	vpmovzxwd Vx, Ux/Mq	vpmovzxwq Vx, Ux/Md	vpmovzxdq Vx, Ux/Mq	vpermd ^Y Vqq, Hqq, Wqq	vpcmpgtq Vx, Hx, Wx
4	vpmulld Vx, Hx, Wx	vphminposuw Vdq, Wdq				vpsrlvd/q ^Y Vx, Hx, Wx	vpsravd ^Y Vx, Hx, Wx	vpsllvd/q ^Y Vx, Hx, Wx
5								
6								
7								
8	INVEPT Gy, Mdq	INVPID Gy, Mdq	INVPCID Gy, Mdq					
9	vgatherdd/q ^Y Vx,Hx,Wx	vgatherqd/q ^Y Vx,Hx,Wx	vgatherdps/d ^Y Vx,Hx,Wx	vgatherqps/d ^Y Vx,Hx,Wx			vfmaddsub132ps/ d ^Y Vx,Hx,Wx	vfmsubadd132ps/ d ^Y Vx,Hx,Wx
A							vfmaddsub213ps/ d ^Y Vx,Hx,Wx	vfmsubadd213ps/ d ^Y Vx,Hx,Wx
B							vfmaddsub231ps/ d ^Y Vx,Hx,Wx	vfmsubadd231ps/ d ^Y Vx,Hx,Wx
C								
D								
E								
F		MOVBE Gy, My	MOVBE My, Gy	ANDN ^Y Gy, By, Ey	Grp 17 ^{1A}		BZHI ^Y Gy, Ey, By	BEXTR ^Y Gy, Ey, By
	66	MOVBE Gw, Mw	MOVBE Mw, Gw					SHLX ^Y Gy, Ey, By
	F3						PEXT ^Y Gy, By, Ey	SARX ^Y Gy, Ey, By
	F2	CRC32 Gd, Eb	CRC32 Gd, Ey				PDEP ^Y Gy, By, Ey	MULX ^Y By,Gy,rDX,Ey
66 & F2	CRC32 Gd, Eb	CRC32 Gd, Ew						

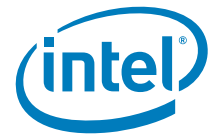


Table A-4. Three-byte Opcode Map: 08H – FFH (First Two Bytes are 0F 38H) *

	px	8	9	A	B	C	D	E	F
0		psignb Pq, Qq	psignw Pq, Qq	psignd Pq, Qq	pmulhrsw Pq, Qq				
	66	vpsignb Vx, Hx, Wx	vpsignw Vx, Hx, Wx	vpsignd Vx, Hx, Wx	vpmulhrsw Vx, Hx, Wx	vpermilps ^v Vx,Hx,Wx	vpermilpd ^v Vx,Hx,Wx	vtestps ^v Vx, Wx	vtestpd ^v Vx, Wx
1						pabsb Pq, Qq	pabsw Pq, Qq	pabsd Pq, Qq	
	66	vbroadcastss ^v Vx, Wd	vbroadcastsd ^v Vqq, Wq	vbroadcastf128 ^v Vqq, Mdq		vpabsb Vx, Wx	vpabsw Vx, Wx	vpabsd Vx, Wx	
2	66	vpmuldq Vx, Hx, Wx	vpcmpeqq Vx, Hx, Wx	vmovntdqa Vx, Mx	vpackusdw Vx, Hx, Wx	vmaskmovps ^v Vx,Hx,Mx	vmaskmovpd ^v Vx,Hx,Mx	vmaskmovps ^v Mx,Hx,Vx	vmaskmovpd ^v Mx,Hx,Vx
3	66	vpminsb Vx, Hx, Wx	vpminsd Vx, Hx, Wx	vpminuw Vx, Hx, Wx	vpminud Vx, Hx, Wx	vpmaxsb Vx, Hx, Wx	vpmaxsd Vx, Hx, Wx	vpmaxuw Vx, Hx, Wx	vpmaxud Vx, Hx, Wx
4									
5	66	vpbroadcastd ^v Vx, Wx	vpbroadcastq ^v Vx, Wx	vpbroadcastf128 ^v Vqq, Mdq					
6									
7	66	vpbroadcastb ^v Vx, Wx	vpbroadcastw ^v Vx, Wx						
8	66					vpmaskmovd/q ^v Vx,Hx,Mx		vpmaskmovd/q ^v Mx,Vx,Hx	
9	66	vfmadd132ps/d ^v Vx, Hx, Wx	vfmadd132ss/d ^v Vx, Hx, Wx	vfmsub132ps/d ^v Vx, Hx, Wx	vfmsub132ss/d ^v Vx, Hx, Wx	vfnmadd132ps/d ^v Vx, Hx, Wx	vfnmadd132ss/d ^v Vx, Hx, Wx	vfnmsub132ps/d ^v Vx, Hx, Wx	vfnmsub132ss/d ^v Vx, Hx, Wx
A	66	vfmadd213ps/d ^v Vx, Hx, Wx	vfmadd213ss/d ^v Vx, Hx, Wx	vfmsub213ps/d ^v Vx, Hx, Wx	vfmsub213ss/d ^v Vx, Hx, Wx	vfnmadd213ps/d ^v Vx, Hx, Wx	vfnmadd213ss/d ^v Vx, Hx, Wx	vfnmsub213ps/d ^v Vx, Hx, Wx	vfnmsub213ss/d ^v Vx, Hx, Wx
B	66	vfmadd231ps/d ^v Vx, Hx, Wx	vfmadd231ss/d ^v Vx, Hx, Wx	vfmsub231ps/d ^v Vx, Hx, Wx	vfmsub231ss/d ^v Vx, Hx, Wx	vfnmadd231ps/d ^v Vx, Hx, Wx	vfnmadd231ss/d ^v Vx, Hx, Wx	vfnmsub231ps/d ^v Vx, Hx, Wx	vfnmsub231ss/d ^v Vx, Hx, Wx
C									
D	66				VAESIMC Vdq, Wdq	VAESEN Vdq,Hdq,Wdq	VAESENCLAST Vdq,Hdq,Wdq	VAESDEC Vdq,Hdq,Wdq	VAESDECLAST Vdq,Hdq,Wdq
E									
F	66								
	F3								
	F2								
	66 & 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.

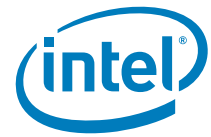


Table A-5 Three-byte Opcode Map: 00H – F7H (First two bytes are 0F 3AH) *

	px	0	1	2	3	4	5	6	7
0	66	vpermq ^v Vqq, Wqq, lb	vpermpd ^v Vqq, Wqq, lb	vpblendd ^v Vx,Hx,Wx,lb		vpermilps ^v Vx, Wx, lb	vpermilpd ^v Vx, Wx, lb	vperm2f128 ^v Vqq,Hqq,Wqq,lb	
1	66					vpextrb Rd/Mb, Vdq, lb	vpextrw Rd/Mw, Vdq, lb	vpextrd/q Ey, Vdq, lb	vextractps Ed, Vdq, lb
2	66	vpinsrb Vdq,Hdq, Ry/ Mb,lb	vinsertps Vdq,Hdq, Udq/ Md,lb	vpinsrd/q Vdq,Hdq,Ey,lb					
3									
4	66	vdpps Vx,Hx,Wx,lb	vdppd Vdq,Hdq,Wdq,lb	vmpsadbw Vx,Hx,Wx,lb		vpcmulqdq Vdq,Hdq,Wdq,lb		vperm2i128 ^v Vqq,Hqq,Wqq,lb	
5									
6	66	vpcmpstrm Vdq, Wdq, lb	vpcmpstri Vdq, Wdq, lb	vpcmpistrm Vdq, Wdq, lb	vpcmpistri Vdq, Wdq, lb				
7									
8									
9									
A									
B									
C									
D									
E									
F	F2	RORX ^v Gy, Ey, lb							

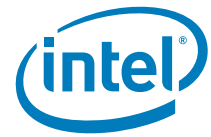


Table A-5. Three-byte Opcode Map: 08H – FFH (First Two Bytes are 0F 3AH) *

	pxf	8	9	A	B	C	D	E	F
0									palignr Pq, Qq, Ib
	66	vroundps Vx,Wx,Ib	vroundpd Vx,Wx,Ib	vroundss Vss,Wss,Ib	vroundsd Vsd,Wsd,Ib	vblendps Vx,Hx,Wx,Ib	vblendpd Vx,Hx,Wx,Ib	vpblendw Vx,Hx,Wx,Ib	vpalignr Vx,Hx,Wx,Ib
1	66	vinserf128 ^v Vqq,Hqq,Wqq,Ib	vextractf128 ^v Wdq,Vqq,Ib				vcvtps2ph ^v Wx, Vx, Ib		
2									
3	66	vinseri128 ^v Vqq,Hqq,Wqq,Ib	vextracti128 ^v Wdq,Vqq,Ib						
4	66			vblendvps ^v Vx,Hx,Wx,Lx	vblendvpd ^v Vx,Hx,Wx,Lx	vpblendvb ^v Vx,Hx,Wx,Lx			
5									
6									
7									
8									
9									
A									
B									
C									
D	66								VAESKEYGEN 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.

...

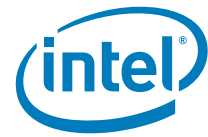


Table A-6 Opcode Extensions for One- and Two-byte Opcodes by Group Number *

Opcode	Group	Mod 7,6	px	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 lb/lz		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 ⁶⁴ Ev	CALLF Ep	JMPN ⁶⁴ Ev	JMPF Mp	PUSH ⁶⁴ 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 SWAPGS ⁶⁴ (000) RDTSCP (001)
		11B	VMCALL (001) VMLAUNCH (010) VMRESUME (011) VMXOFF (100)	MONITOR (000) MWAIT (001)	XGETBV (000) XSETBV (001) VMFUNC (100) XEND (101) XTEST (110)						
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								RDRAND Rv	
0F B9	10	mem 11B									
C6	11	mem		MOV Eb, lb							
		11B									XABORT (000) lb
C7	11	mem		MOV Ev, lz							
		11B									XBEGIN (000) Jz



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
OF 71	12	mem									
		11B			psrlw Nq, lb		psraw Nq, lb		psllw Nq, lb		
			66			vpsrlw Hx,Ux,lb		vpsraw Hx,Ux,lb		vpsllw Hx,Ux,lb	
OF 72	13	mem									
		11B			psrld Nq, lb		psrad Nq, lb		pslld Nq, lb		
			66			vpsrld Hx,Ux,lb		vpsrad Hx,Ux,lb		vpslld Hx,Ux,lb	
OF 73	14	mem									
		11B			psrlq Nq, lb				psllq Nq, lb		
			66			vpsrlq Hx,Ux,lb	vpsrlq Hx,Ux,lb		vpsllq Hx,Ux,lb	vpslldq Hx,Ux,lb	
OF AE	15	mem		fxsave	fxrstor	ldmxcsr	stmxcsr	XSAVE	XRSTOR	XSAVEOPT	clflush
		11B	F3	RDFSBASE Ry	RDGSBASE Ry	WRFSBASE Ry	WRGSBASE Ry		lfence	mfence	sfence
OF 18	16	mem		prefetch NTA	prefetch T0	prefetch T1	prefetch T2				
		11B									
VEX.0F38 F3	17	mem			BLSR ^v By, Ey	BLMSK ^v By, Ey	BLSI ^v By, Ey				
		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.

...

6. Updates to Appendix B, Volume 2C

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

...

B.2 GENERAL-PURPOSE INSTRUCTION FORMATS AND ENCODINGS FOR NON-64-BIT MODES

Table B-13 shows machine instruction formats and encodings for general purpose instructions in non-64-bit modes.

Table B-13 General Purpose Instruction Formats and Encodings for Non-64-Bit Modes

Instruction and Format	Encoding
...	
MOV - Move Data	

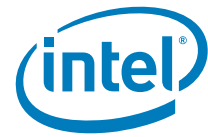


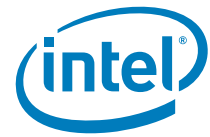
Table B-13 General Purpose Instruction Formats and Encodings for Non-64-Bit Modes (Contd.)

Instruction and Format	Encoding
register1 to register2	1000 100w : 11 reg1 reg2
register2 to register1	1000 101w : 11 reg1 reg2
memory to reg	1000 101w : mod reg r/m
reg to memory	1000 100w : mod reg r/m
immediate to register	1100 011w : 11 000 reg : immediate data
immediate to register (alternate encoding)	1011 w reg : immediate data
immediate to memory	1100 011w : mod 000 r/m : immediate data
memory to AL, AX, or EAX	1010 000w : full displacement
AL, AX, or EAX to memory	1010 001w : full displacement
MOV - Move to/from Control Registers	
CR0 from register	0000 1111 : 0010 0010 : -- 000 reg
CR2 from register	0000 1111 : 0010 0010 : -- 010reg
CR3 from register	0000 1111 : 0010 0010 : -- 011 reg
CR4 from register	0000 1111 : 0010 0010 : -- 100 reg
register from CR0-CR4	0000 1111 : 0010 0000 : -- eee reg
MOV - Move to/from Debug Registers	
DR0-DR3 from register	0000 1111 : 0010 0011 : -- eee reg
DR4-DR5 from register	0000 1111 : 0010 0011 : -- eee reg
DR6-DR7 from register	0000 1111 : 0010 0011 : -- eee reg
register from DR6-DR7	0000 1111 : 0010 0001 : -- eee reg
register from DR4-DR5	0000 1111 : 0010 0001 : -- eee reg
register from DR0-DR3	0000 1111 : 0010 0001 : -- eee reg
MOV - Move to/from Segment Registers	
register to segment register	1000 1110 : 11 sreg3 reg
register to SS	1000 1110 : 11 sreg3 reg
memory to segment reg	1000 1110 : mod sreg3 r/m
memory to SS	1000 1110 : mod sreg3 r/m
segment register to register	1000 1100 : 11 sreg3 reg
segment register to memory	1000 1100 : mod sreg3 r/m
...	

...

7. Updates to Chapter 1, Volume 3A

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

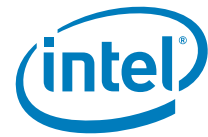


...

1.1 INTEL® 64 AND IA-32 PROCESSORS COVERED IN THIS MANUAL

This manual set includes information pertaining primarily to the most recent Intel 64 and IA-32 processors, which include:

- Pentium® processors
- P6 family processors
- Pentium® 4 processors
- Pentium® M processors
- Intel® Xeon® processors
- Pentium® D processors
- Pentium® processor Extreme Editions
- 64-bit Intel® Xeon® processors
- Intel® Core™ Duo processor
- Intel® Core™ Solo processor
- Dual-Core Intel® Xeon® processor LV
- Intel® Core™2 Duo processor
- Intel® Core™2 Quad processor Q6000 series
- Intel® Xeon® processor 3000, 3200 series
- Intel® Xeon® processor 5000 series
- Intel® Xeon® processor 5100, 5300 series
- Intel® Core™2 Extreme processor X7000 and X6800 series
- Intel® Core™2 Extreme QX6000 series
- Intel® Xeon® processor 7100 series
- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® Core™2 Extreme QX9000 series
- Intel® Xeon® processor 5200, 5400, 7400 series
- Intel® Core™2 Extreme processor QX9000 and X9000 series
- Intel® Core™2 Quad processor Q9000 series
- Intel® Core™2 Duo processor E8000, T9000 series
- Intel® Atom™ processor family
- Intel® Core™ i7 processor
- Intel® Core™ i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Xeon® processor E5 family
- Intel® Xeon® processor E3 family



- Intel® Core™ i7-3930K processor
- 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series

P6 family processors are IA-32 processors based on the P6 family microarchitecture. This includes the Pentium® Pro, Pentium® II, Pentium® III, and Pentium® III Xeon® processors.

The Pentium® 4, Pentium® D, and Pentium® processor Extreme Editions are based on the Intel NetBurst® microarchitecture. Most early Intel® Xeon® processors are based on the Intel NetBurst® microarchitecture. Intel Xeon processor 5000, 7100 series are based on the Intel NetBurst® microarchitecture.

The Intel® Core™ Duo, Intel® Core™ Solo and dual-core Intel® Xeon® processor LV are based on an improved Pentium® M processor microarchitecture.

The Intel® Xeon® processor 3000, 3200, 5100, 5300, 7200, and 7300 series, Intel® Pentium® dual-core, Intel® Core™2 Duo, Intel® Core™2 Quad and Intel® Core™2 Extreme processors are based on Intel® Core™ microarchitecture.

The Intel® Xeon® processor 5200, 5400, 7400 series, Intel® Core™2 Quad processor Q9000 series, and Intel® Core™2 Extreme processors QX9000, X9000 series, Intel® Core™2 processor E8000 series are based on Enhanced Intel® Core™ microarchitecture.

The Intel® Atom™ processor family is based on the Intel® Atom™ microarchitecture and supports Intel 64 architecture.

The Intel® Core™ i7 processor and the Intel® Core™ i5 processor are based on the Intel® microarchitecture code name Nehalem and support Intel 64 architecture.

Processors based on Intel® microarchitecture code name Westmere support Intel 64 architecture.

P6 family, Pentium® M, Intel® Core™ Solo, Intel® Core™ Duo processors, dual-core Intel® Xeon® processor LV, and early generations of Pentium 4 and Intel Xeon processors support IA-32 architecture. The Intel® Atom™ processor Z5xx series support IA-32 architecture.

The Intel® Xeon® processor E5 family, Intel® Xeon® processor E3 family, Intel® Core™ i7-3930K processor, 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series, Intel® Xeon® processor E7-8800/4800/2800 product families, Intel® Xeon® processor 3000, 3200, 5000, 5100, 5200, 5300, 5400, 7100, 7200, 7300, 7400 series, Intel® Core™2 Duo, Intel® Core™2 Extreme processors, Intel Core 2 Quad processors, Pentium® D processors, Pentium® Dual-Core processor, newer generations of Pentium 4 and Intel Xeon processor family support Intel® 64 architecture.

IA-32 architecture is the instruction set architecture and programming environment for Intel's 32-bit microprocessors. Intel® 64 architecture is the instruction set architecture and programming environment which is a superset of and compatible with IA-32 architecture.

...

8. 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*.

...

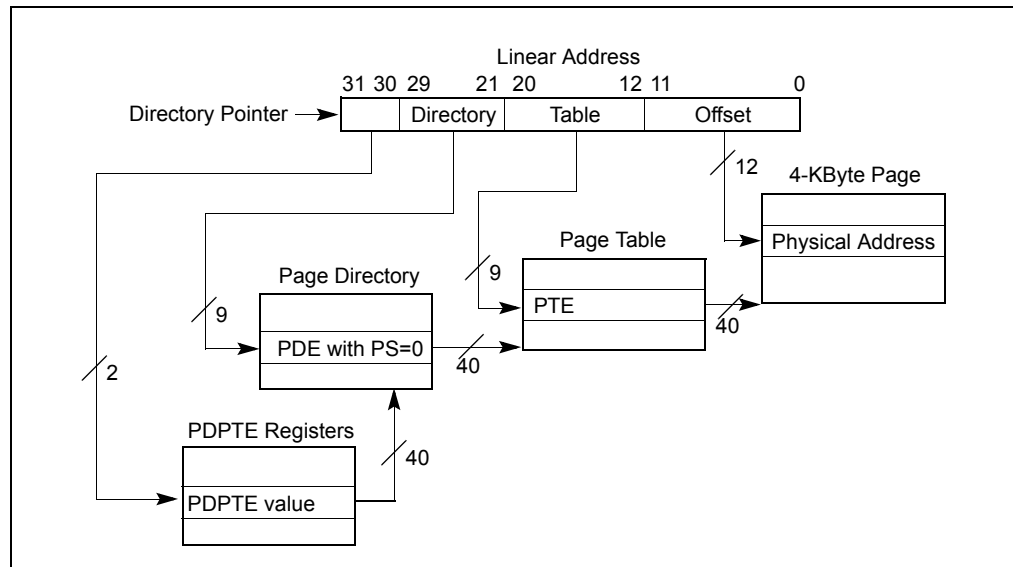


Figure 4-5 Linear-Address Translation to a 4-KByte Page using PAE Paging

...

9. Updates to Chapter 10, Volume 3A

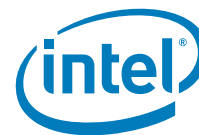
Change bars show changes to Chapter 10 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

...

10.5.1 Local Vector Table

The local vector table (LVT) allows software to specify the manner in which the local interrupts are delivered to the processor core. It consists of the following 32-bit APIC registers (see Figure 10-8), one for each local interrupt:

- **LVT CMCI Register (FEE0 02F0H)** — Specifies interrupt delivery when an overflow condition of corrected machine check error count reaching a threshold value occurred in a machine check bank supporting CMCI (see Section 15.5.1, "CMCI Local APIC Interface").
- **LVT Timer Register (FEE0 0320H)** — Specifies interrupt delivery when the APIC timer signals an interrupt (see Section 10.5.4, "APIC Timer").
- **LVT Thermal Monitor Register (FEE0 0330H)** — Specifies interrupt delivery when the thermal sensor generates an interrupt (see Section 14.5.2, "Thermal Monitor"). This LVT entry is implementation specific, not architectural. If implemented, it will always be at base address FEE0 0330H.
- **LVT Performance Counter Register (FEE0 0340H)** — Specifies interrupt delivery when a performance counter generates an interrupt on overflow (see



Section 18.10.5.8, “Generating an Interrupt on Overflow”). This LVT entry is implementation specific, not architectural. If implemented, it is not guaranteed to be at base address FEE0 0340H.

- **LVT LINT0 Register (FEE0 0350H)** — Specifies interrupt delivery when an interrupt is signaled at the LINT0 pin.
- **LVT LINT1 Register (FEE0 0360H)** — Specifies interrupt delivery when an interrupt is signaled at the LINT1 pin.
- **LVT Error Register (FEE0 0370H)** — Specifies interrupt delivery when the APIC detects an internal error (see Section 10.5.3, “Error Handling”).

The LVT performance counter register and its associated interrupt were introduced in the P6 processors and are also present in the Pentium 4 and Intel Xeon processors. The LVT thermal monitor register and its associated interrupt were introduced in the Pentium 4 and Intel Xeon processors. The LVT CMCI register and its associated interrupt were introduced in the Intel Xeon 5500 processors.

As shown in Figure 10-8, some of these fields and flags are not available (and reserved) for some entries.

The setup information that can be specified in the registers of the LVT table is as follows:

Vector	Interrupt vector number.
Delivery Mode	Specifies the type of interrupt to be sent to the processor. Some delivery modes will only operate as intended when used in conjunction with a specific trigger mode. The allowable delivery modes are as follows:
000 (Fixed)	Delivers the interrupt specified in the vector field.
010 (SMI)	Delivers an SMI interrupt to the processor core through the processor’s local SMI signal path. When using this delivery mode, the vector field should be set to 00H for future compatibility.
100 (NMI)	Delivers an NMI interrupt to the processor. The vector information is ignored.
101 (INIT)	Delivers an INIT request to the processor core, which causes the processor to perform an INIT. When using this delivery mode, the vector field should be set to 00H for future compatibility. Not supported for the LVT CMCI register, the LVT thermal monitor register, or the LVT performance counter register.
110	Reserved; not supported for any LVT register.
111 (ExtINT)	Causes the processor to respond to the interrupt as if the interrupt originated in an external-

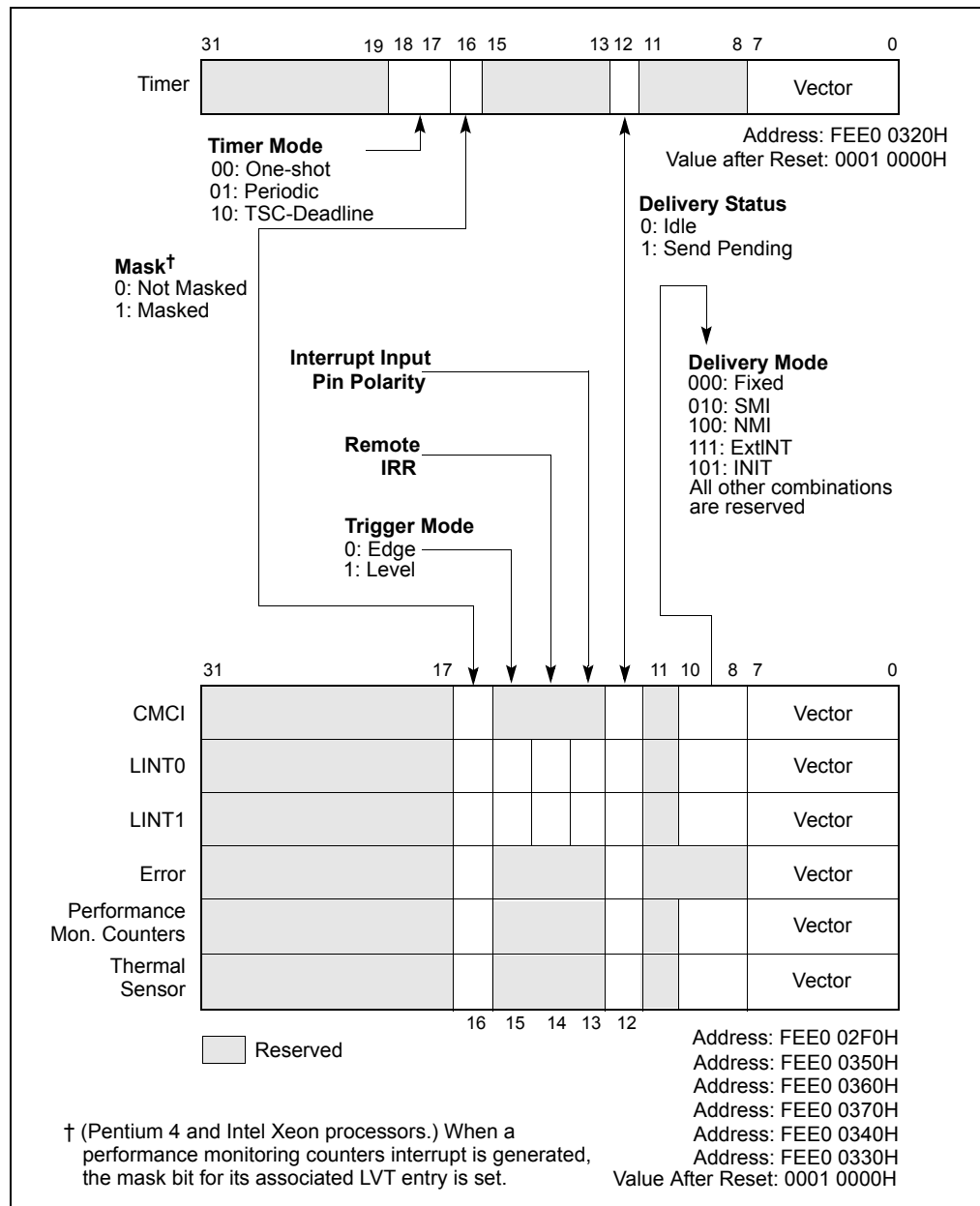


Figure 10-8 Local Vector Table (LVT)

ly connected (8259A-compatible) interrupt controller. A special INTA bus cycle corresponding to ExtINT, is routed to the external controller. The external controller is expected to supply the vector information. The APIC architecture supports only one ExtINT source in a system, usually contained in the compatibility bridge. Only one processor in the system should have an LVT entry configured to use the ExtINT deliv-



ery mode. Not supported for the LVT CMCI register, the LVT thermal monitor register, or the LVT performance counter register.

Delivery Status (Read Only)

Indicates the interrupt delivery status, as follows:

0 (Idle) There is currently no activity for this interrupt source, or the previous interrupt from this source was delivered to the processor core and accepted.

1 (Send Pending) Indicates that an interrupt from this source has been delivered to the processor core but has not yet been accepted (see Section 10.5.5, “Local Interrupt Acceptance”).

Interrupt Input Pin Polarity

Specifies the polarity of the corresponding interrupt pin: (0) active high or (1) active low.

Remote IRR Flag (Read Only)

For fixed mode, level-triggered interrupts; this flag is set when the local APIC accepts the interrupt for servicing and is reset when an EOI command is received from the processor. The meaning of this flag is undefined for edge-triggered interrupts and other delivery modes.

Trigger Mode

Selects the trigger mode for the local LINT0 and LINT1 pins: (0) edge sensitive and (1) level sensitive. This flag is only used when the delivery mode is Fixed. When the delivery mode is NMI, SMI, or INIT, the trigger mode is always edge sensitive. When the delivery mode is ExtINT, the trigger mode is always level sensitive. The timer and error interrupts are always treated as edge sensitive.

If the local APIC is not used in conjunction with an I/O APIC and fixed delivery mode is selected; the Pentium 4, Intel Xeon, and P6 family processors will always use level-sensitive triggering, regardless if edge-sensitive triggering is selected.

Mask

Interrupt mask: (0) enables reception of the interrupt and (1) inhibits reception of the interrupt. When the local APIC handles a performance-monitoring counters interrupt, it automatically sets the mask flag in the LVT performance counter register. This flag is set to 1 on reset. It can be cleared only by software.

Timer Mode

Bits 18:17 selects the timer mode (see Section 10.5.4):
(00b) one-shot mode using a count-down value,
(01b) periodic mode reloading a count-down value,
(10b) TSC-Deadline mode using absolute target value in IA32_TSC_DEADLINE MSR (see Section 10.5.4.1),
(11b) is reserved.

...

10.8.1 Interrupt Handling with the Pentium 4 and Intel Xeon Processors

With the Pentium 4 and Intel Xeon processors, the local APIC handles the local interrupts, interrupt messages, and IPIs it receives as follows:

1. It determines if it is the specified destination or not (see Figure 10-16). If it is the specified destination, it accepts the message; if it is not, it discards the message.

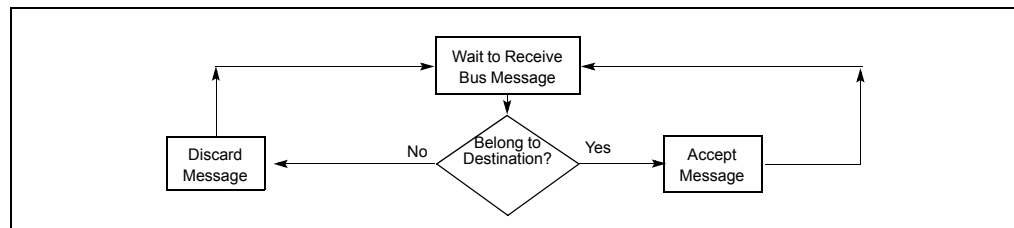


Figure 10-16 Interrupt Acceptance Flow Chart for the Local APIC (Pentium 4 and Intel Xeon Processors)

2. If the local APIC determines that it is the designated destination for the interrupt and if the interrupt request is an NMI, SMI, INIT, ExtINT, or SIPI, the interrupt is sent directly to the processor core for handling.
3. If the local APIC determines that it is the designated destination for the interrupt but the interrupt request is not one of the interrupts given in step 2, the local APIC sets the appropriate bit in the IRR.
4. When interrupts are pending in the IRR register, the local APIC dispatches them to the processor one at a time, based on their priority and the current processor priority in the PPR (see Section 10.8.3.1, “Task and Processor Priorities”).
5. When a fixed interrupt has been dispatched to the processor core for handling, the completion of the handler routine is indicated with an instruction in the instruction handler code that writes to the end-of-interrupt (EOI) register in the local APIC (see Section 10.8.5, “Signaling Interrupt Servicing Completion”). The act of writing to the EOI register causes the local APIC to delete the interrupt from its ISR queue and (for level-triggered interrupts) send a message on the bus indicating that the interrupt handling has been completed. (A write to the EOI register must not be included in the handler routine for an NMI, SMI, INIT, ExtINT, or SIPI.)

...

10.8.2 Interrupt Handling with the P6 Family and Pentium Processors

With the P6 family and Pentium processors, the local APIC handles the local interrupts, interrupt messages, and IPIs it receives as follows (see Figure 10-17).

1. (IPIs only) It examines the IPI message to determine if it is the specified destination for the IPI as described in Section 10.6.2, “Determining IPI Destination.” If it is the specified destination, it continues its acceptance procedure; if it is not the destination, it discards the IPI message. When the message specifies lowest-priority delivery mode, the local APIC will arbitrate with the other processors that were designated on recipients of the IPI message (see Section 10.6.2.4, “Lowest Priority Delivery Mode”).

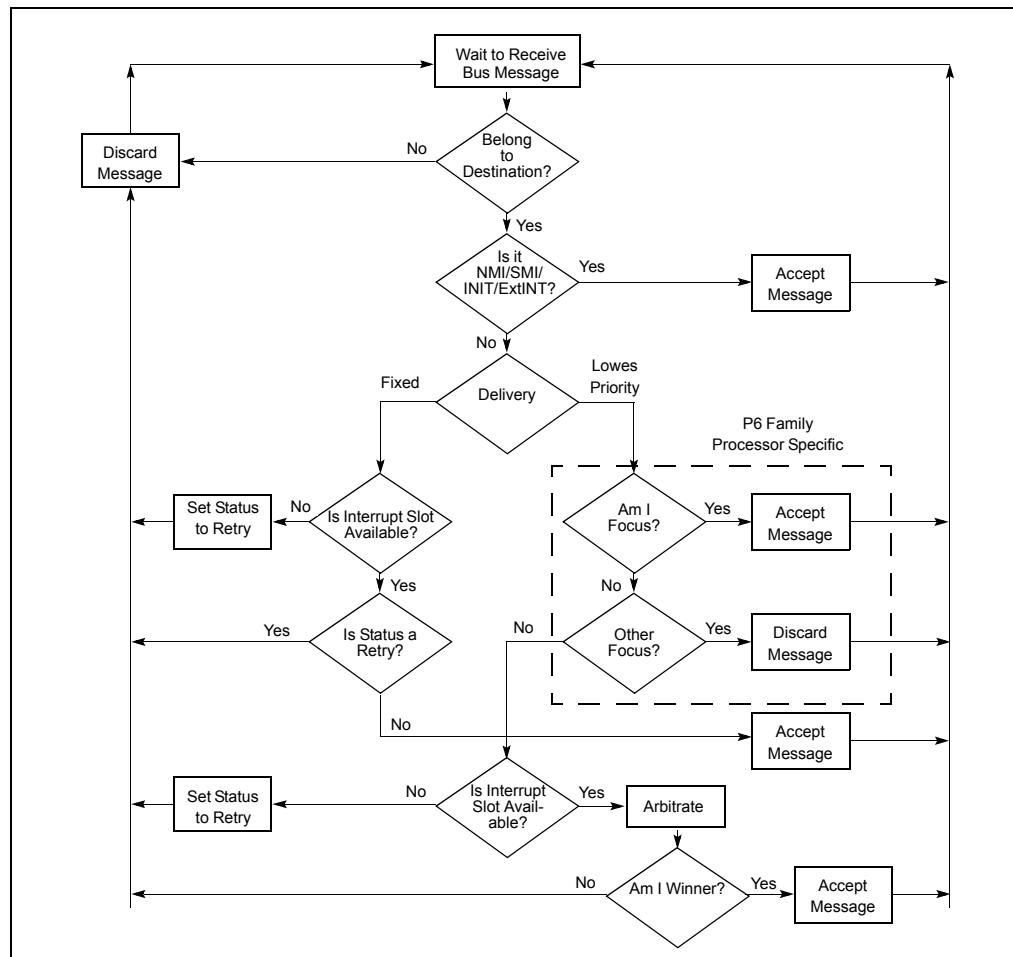
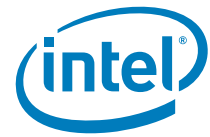


Figure 10-17 Interrupt Acceptance Flow Chart for the Local APIC (P6 Family and Pentium Processors)

2. If the local APIC determines that it is the designated destination for the interrupt and if the interrupt request is an NMI, SMI, INIT, ExtINT, or INIT-deassert interrupt, or one of the MP protocol IPI messages (BIPI, FIPI, and SIPI), the interrupt is sent directly to the processor core for handling.
3. If the local APIC determines that it is the designated destination for the interrupt but the interrupt request is not one of the interrupts given in step 2, the local APIC looks for an open slot in one of its two pending interrupt queues contained in the IRR and ISR registers (see Figure 10-20). If a slot is available (see Section 10.8.4, "Interrupt Acceptance for Fixed Interrupts"), places the interrupt in the slot. If a slot is not available, it rejects the interrupt request and sends it back to the sender with a retry message.
4. When interrupts are pending in the IRR register, the local APIC dispatches them to the processor one at a time, based on their priority and the current processor priority in the PPR (see Section 10.8.3.1, "Task and Processor Priorities").
5. When a fixed interrupt has been dispatched to the processor core for handling, the completion of the handler routine is indicated with an instruction in the instruction



handler code that writes to the end-of-interrupt (EOI) register in the local APIC (see Section 10.8.5, “Signaling Interrupt Servicing Completion”). The act of writing to the EOI register causes the local APIC to delete the interrupt from its queue and (for level-triggered interrupts) send a message on the bus indicating that the interrupt handling has been completed. (A write to the EOI register must not be included in the handler routine for an NMI, SMI, INIT, ExtINT, or SIPI.)

The following sections describe the acceptance of interrupts and their handling by the local APIC and processor in greater detail.

...

10.8.3 Interrupt, Task, and Processor Priority

Each interrupt delivered to the processor through the local APIC has a priority based on its vector number. The local APIC uses this priority to determine when to service the interrupt relative to the other activities of the processor, including the servicing of other interrupts.

Each interrupt vector is an 8-bit value. The **interrupt-priority class** is the value of bits 7:4 of the interrupt vector. The lowest interrupt-priority class is 1 and the highest is 15; interrupts with vectors in the range 0–15 (with interrupt-priority class 0) are illegal and are never delivered. Because vectors 0–31 are reserved for dedicated uses by the Intel 64 and IA-32 architectures, software should configure interrupt vectors to use interrupt-priority classes in the range 2–15.

Each interrupt-priority class encompasses 16 vectors. The relative priority of interrupts within an interrupt-priority class is determined by the value of bits 3:0 of the vector number. The higher the value of those bits, the higher the priority within that interrupt-priority class. Thus, each interrupt vector comprises two parts, with the high 4 bits indicating its interrupt-priority class and the low 4 bits indicating its ranking within the interrupt-priority class.

10.8.3.1 Task and Processor Priorities

The local APIC also defines a **task priority** and a **processor priority** that determine the order in which interrupts are handled. The **task-priority class** is the value of bits 7:4 of the task-priority register (TPR), which can be written by software (TPR is a read/write register); see Figure 10-18.

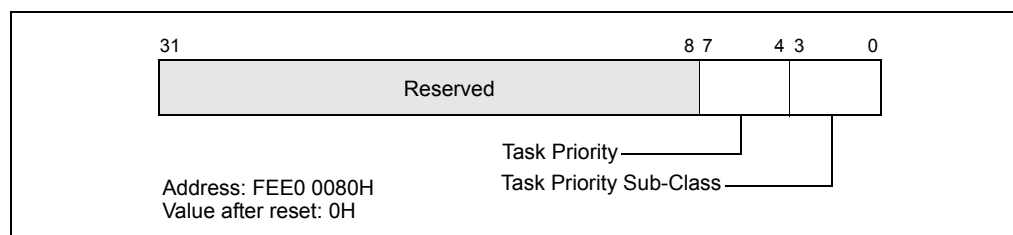


Figure 10-18 Task-Priority Register (TPR)

NOTE

In this discussion, the term “task” refers to a software defined task, process, thread, program, or routine that is dispatched to run on the

processor by the operating system. It does not refer to an IA-32 architecture defined task as described in Chapter 7, “Task Management.”

The task priority allows software to set a priority threshold for interrupting the processor. This mechanism enables the operating system to temporarily block low priority interrupts from disturbing high-priority work that the processor is doing. The ability to block such interrupts using task priority results from the way that the TPR controls the value of the processor-priority register (PPR).¹

The **processor-priority class** is a value in the range 0–15 that is maintained in bits 7:4 of the processor-priority register (PPR); see Figure 10-19. The PPR is a read-only register. The processor-priority class represents the current priority at which the processor is executing.

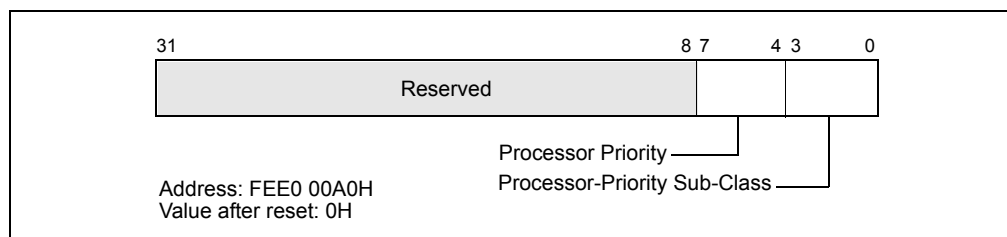


Figure 10-19 Processor-Priority Register (PPR)

The value of the PPR is based on the value of TPR and the value ISRV; ISRV is the vector number of the highest priority bit that is set in the ISR or 00H if no bit is set in the ISR. (See Section 10.8.4 for more details on the ISR.) The value of PPR is determined as follows:

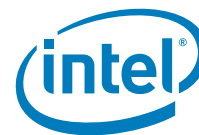
- PPR[7:4] (the processor-priority class) the maximum of TPR[7:4] (the task-priority class) and ISRV[7:4] (the priority of the highest priority interrupt in service).
- PPR[3:0] (the processor-priority sub-class) is determined as follows:
 - If TPR[7:4] > ISRV[7:4], PPR[3:0] is TPR[3:0] (the task-priority sub-class).
 - If TPR[7:4] < ISRV[7:4], PPR[3:0] is 0.
 - If TPR[7:4] = ISRV[7:4], PPR[3:0] may be either TPR[3:0] or 0. The actual behavior is model-specific.

The processor-priority class determines the priority threshold for interrupting the processor. The processor will deliver only those interrupts that have an interrupt-priority class higher than the processor-priority class in the PPR. If the processor-priority class is 0, the PPR does not inhibit the delivery any interrupt; if it is 15, the processor inhibits the delivery of all interrupts. (The processor-priority mechanism does not affect the delivery of interrupts with the NMI, SMI, INIT, ExtINT, INIT-deassert, and start-up delivery modes.)

The processor does not use the processor-priority sub-class to determine which interrupts to delivery and which to inhibit. (The processor uses the processor-priority sub-class only to satisfy reads of the PPR.)

...

1. The TPR also determines the arbitration priority of the local processor; see Section 10.6.2.4, “Lowest Priority Delivery Mode.”



10.8.4 Interrupt Acceptance for Fixed Interrupts

The local APIC queues the fixed interrupts that it accepts in one of two interrupt pending registers: the interrupt request register (IRR) or in-service register (ISR). These two 256-bit read-only registers are shown in Figure 10-20. The 256 bits in these registers represent the 256 possible vectors; vectors 0 through 15 are reserved by the APIC (see also: Section 10.5.2, "Valid Interrupt Vectors").

NOTE

All interrupts with an NMI, SMI, INIT, ExtINT, start-up, or INIT-deassert delivery mode bypass the IRR and ISR registers and are sent directly to the processor core for servicing.

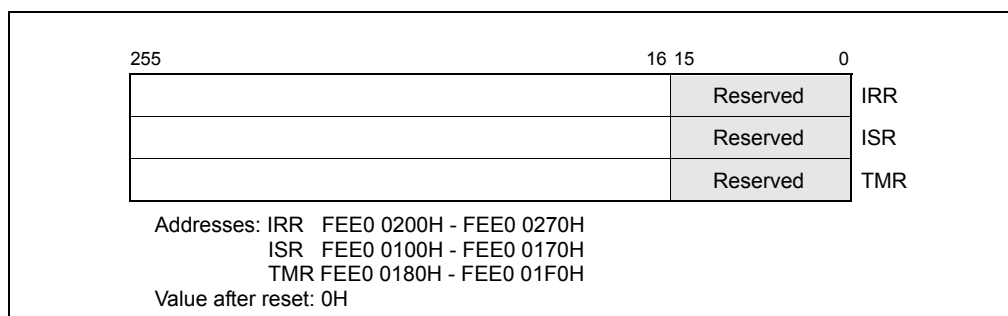


Figure 10-20 IRR, ISR and TMR Registers

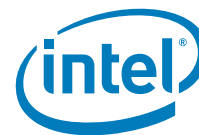
The IRR contains the active interrupt requests that have been accepted, but not yet dispatched to the processor for servicing. When the local APIC accepts an interrupt, it sets the bit in the IRR that corresponds the vector of the accepted interrupt. When the processor core is ready to handle the next interrupt, the local APIC clears the highest priority IRR bit that is set and sets the corresponding ISR bit. The vector for the highest priority bit set in the ISR is then dispatched to the processor core for servicing.

While the processor is servicing the highest priority interrupt, the local APIC can send additional fixed interrupts by setting bits in the IRR. When the interrupt service routine issues a write to the EOI register (see Section 10.8.5, "Signaling Interrupt Servicing Completion"), the local APIC responds by clearing the highest priority ISR bit that is set. It then repeats the process of clearing the highest priority bit in the IRR and setting the corresponding bit in the ISR. The processor core then begins executing the service routing for the highest priority bit set in the ISR.

If more than one interrupt is generated with the same vector number, the local APIC can set the bit for the vector both in the IRR and the ISR. This means that for the Pentium 4 and Intel Xeon processors, the IRR and ISR can queue two interrupts for each interrupt vector: one in the IRR and one in the ISR. Any additional interrupts issued for the same interrupt vector are collapsed into the single bit in the IRR.

For the P6 family and Pentium processors, the IRR and ISR registers can queue no more than two interrupts per interrupt vector and will reject other interrupts that are received within the same vector.

If the local APIC receives an interrupt with an interrupt-priority class higher than that of the interrupt currently in service, and interrupts are enabled in the processor core, the local APIC dispatches the higher priority interrupt to the processor immediately (without waiting for a write to the EOI register). The currently executing interrupt handler is then interrupted so the higher-priority interrupt can be handled. When the handling of the



higher-priority interrupt has been completed, the servicing of the interrupted interrupt is resumed.

The trigger mode register (TMR) indicates the trigger mode of the interrupt (see Figure 10-20). Upon acceptance of an interrupt into the IRR, the corresponding TMR bit is cleared for edge-triggered interrupts and set for level-triggered interrupts. If a TMR bit is set when an EOI cycle for its corresponding interrupt vector is generated, an EOI message is sent to all I/O APICs.

...

10.8.6 Task Priority in IA-32e Mode

In IA-32e mode, operating systems can manage the 16 interrupt-priority classes (see Section 10.8.3, "Interrupt, Task, and Processor Priority") explicitly using the task priority register (TPR). Operating systems can use the TPR to temporarily block specific (low-priority) interrupts from interrupting a high-priority task. This is done by loading TPR with a value in which the task-priority class corresponds to the highest interrupt-priority class that is to be blocked. For example:

- Loading the TPR with a task-priority class of 8 (01000B) blocks all interrupts with an interrupt-priority class of 8 or less while allowing all interrupts with an interrupt-priority class of 9 or more to be recognized.
- Loading the TPR with a task-priority class of 0 enables all external interrupts.
- Loading the TPR with a task-priority class of 0FH (01111B) disables all external interrupts.

The TPR (shown in Figure 10-18) is cleared to 0 on reset. In 64-bit mode, software can read and write the TPR using an alternate interface, MOV CR8 instruction. The new task-priority class is established when the MOV CR8 instruction completes execution. Software does not need to force serialization after loading the TPR using MOV CR8.

Use of the MOV CRn instruction requires a privilege level of 0. Programs running at privilege level greater than 0 cannot read or write the TPR. An attempt to do so causes a general-protection exception. The TPR is abstracted from the interrupt controller (IC), which prioritizes and manages external interrupt delivery to the processor. The IC can be an external device, such as an APIC or 8259. Typically, the IC provides a priority mechanism similar or identical to the TPR. The IC, however, is considered implementation-dependent with the underlying priority mechanisms subject to change. CR8, by contrast, is part of the Intel 64 architecture. Software can depend on this definition remaining unchanged.

...

10. Updates to Chapter 14, Volume 3B

Change bars show changes to Chapter 14 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

...

14.7.3 Package RAPL Domain

The MSR interfaces defined for the package RAPL domain are:

- MSR_PKG_POWER_LIMIT allows software to set power limits for the package and measurement attributes associated with each limit,
- MSR_PKG_ENERGY_STATUS reports measured actual energy usage,
- MSR_PKG_POWER_INFO reports the package power range information for RAPL usage.

MSR_PKG_RAPL_PERF_STATUS can report the performance impact of power limiting, but its availability may be model-specific.

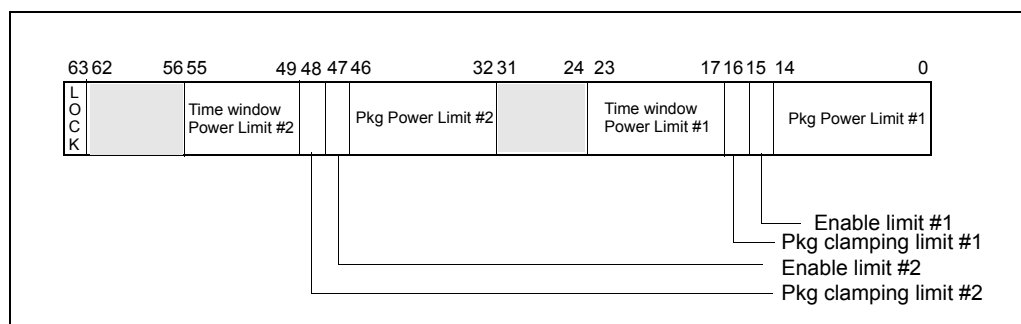


Figure 14-17 MSR_PKG_POWER_LIMIT Register

MSR_PKG_POWER_LIMIT allows a software agent to define power limitation for the package domain. Power limitation is defined in terms of average power usage (Watts) over a time window specified in MSR_PKG_POWER_LIMIT. Two power limits can be specified, corresponding to time windows of different sizes. Each power limit provides independent clamping control that would permit the processor cores to go below OS-requested state to meet the power limits. A lock mechanism allow the software agent to enforce power limit settings. Once the lock bit is set, the power limit settings are static and un-modifiable until next RESET.

The bit fields of MSR_PKG_POWER_LIMIT (Figure 14-17) are:

- **Package Power Limit #1**(bits 14:0): Sets the average power usage limit of the package domain corresponding to time window # 1. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit #1**(bit 15): 0 = disabled; 1 = enabled.
- **Package Clamping Limitation #1** (bit 16): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit #1** (bits 23:17): Indicates the length of time window over which the power limit # 1. The numeric value encoded by bits 23:17 is represented by the product of $2^Y * F$; where F is a single-digit decimal floating-point value between 1.0 and 1.3 with the fraction digit represented by bits 23:22, Y is an unsigned integer represented by bits 21:17. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Package Power Limit #2**(bits 46:32): Sets the average power usage limit of the package domain corresponding to time window # 2. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit #2**(bit 47): 0 = disabled; 1 = enabled.

- **Package Clamping Limitation #2** (bit 48): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit #2** (bits 55:49): Indicates the length of time window over which the power limit #2. The numeric value encoded by bits 55:49 is represented by the product of $2^Y * F$; where F is a single-digit decimal floating-point value between 1.0 and 1.3 with the fraction digit represented by bits 55:54, Y is an unsigned integer represented by bits 53:49. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT. This field may have a hard-coded value in hardware and ignores values written by software.
- **Lock** (bit 63): If set, all write attempts to this MSR are ignored until next RESET.

MSR_PKG_ENERGY_STATUS is a read-only MSR. It reports the actual energy use for the package domain. This MSR is updated every ~ 1 msec. It has a wraparound time of around 60 secs when power consumption is high, and may be longer otherwise.

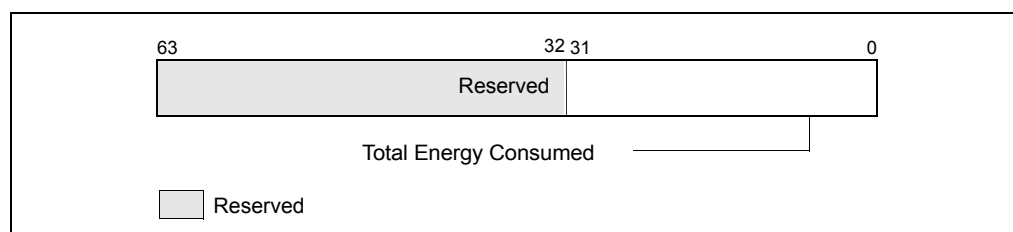


Figure 14-18 MSR_PKG_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since that last time this register is cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_PKG_POWER_INFO is a read-only MSR. It reports the package power range information for RAPL usage. This MSR provides maximum/minimum values (derived from electrical specification), thermal specification power of the package domain. It also provides the largest possible time window for software to program the RAPL interface.

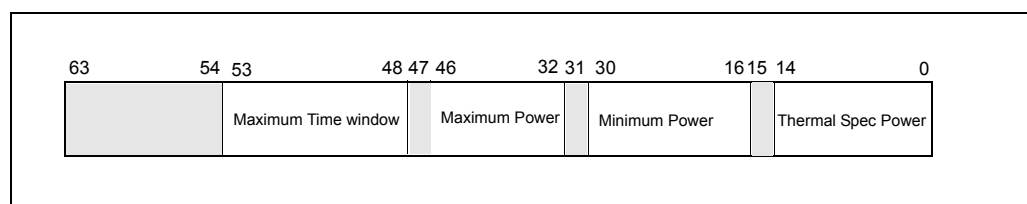


Figure 14-19 MSR_PKG_POWER_INFO Register

- **Thermal Spec Power** (bits 14:0): The unsigned integer value is the equivalent of thermal specification power of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Minimum Power** (bits 30:16): The unsigned integer value is the equivalent of minimum power derived from electrical spec of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.

- **Maximum Power** (bits 46:32): The unsigned integer value is the equivalent of maximum power derived from the electrical spec of the package domain. The unit of this field is specified by the “Power Units” field of MSR_RAPL_POWER_UNIT.
- **Maximum Time Window** (bits 53:48): The unsigned integer value is the equivalent of largest acceptable value to program the time window of MSR_PKG_POWER_LIMIT. The unit of this field is specified by the “Time Units” field of MSR_RAPL_POWER_UNIT.

...

14.7.4 PP0/PP1 RAPL Domains

The MSR interfaces defined for the PP0 and PP1 domains are identical in layout. Generally, PP0 refers to the processor cores. The availability of PP1 RAPL domain interface is platform-specific. For a client platform, PP1 domain refers to the power plane of a specific device in the uncore. For server platforms, PP1 domain is not supported, but its PP0 domain supports the MSR_PP0_PERF_STATUS interface.

- MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT allow software to set power limits for the respective power plane domain.
- MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS report actual energy usage on a power plane.
- MSR_PP0_POLICY/MSR_PP1_POLICY allow software to adjust balance for respective power plane.

MSR_PP0_PERF_STATUS can report the performance impact of power limiting, but it is not available in client platform.

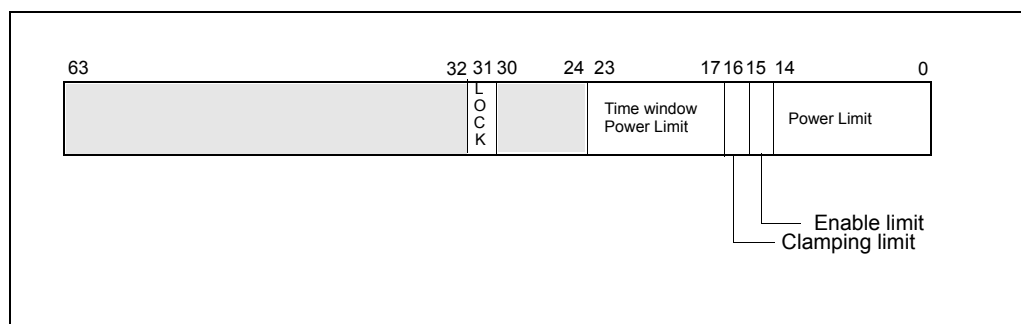
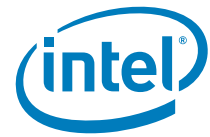


Figure 14-21 MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT Register

MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT allows a software agent to define power limitation for the respective power plane domain. A lock mechanism in each power plane domain allow the software agent to enforce power limit settings independently. Once a lock bit is set, the power limit settings in that power plane are static and unmodifiable until next RESET.

The bit fields of MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT (Figure 14-21) are:

- **Power Limit** (bits 14:0): Sets the average power usage limit of the respective power plane domain. The unit of this field is specified by the “Power Units” field of MSR_RAPL_POWER_UNIT.



- **Enable Power Limit** (bit 15): 0 = disabled; 1 = enabled.
- **Clamping Limitation** (bit 16): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit** (bits 23:17): Indicates the length of time window over which the power limit #1 The numeric value encoded by bits 23:17 is represented by the product of $2^Y * F$; where F is a single-digit decimal floating-point value between 1.0 and 1.3 with the fraction digit represented by bits 23:22, Y is an unsigned integer represented by bits 21:17. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Lock** (bit 31): If set, all write attempts to the MSR and corresponding policy MSR_PP0_POLICY/MSR_PP1_POLICY are ignored until next RESET.

...

14.7.5 DRAM RAPL Domain

The MSR interfaces defined for the DRAM domain is supported only in the server platform. The MSR interfaces are:

- MSR_DRAM_POWER_LIMIT allows software to set power limits for the DRAM domain and measurement attributes associated with each limit,
- MSR_DRAM_ENERGY_STATUS reports measured actual energy usage,
- MSR_DRAM_POWER_INFO reports the DRAM domain power range information for RAPL usage.
- MSR_DRAM_RAPL_PERF_STATUS can report the performance impact of power limiting.

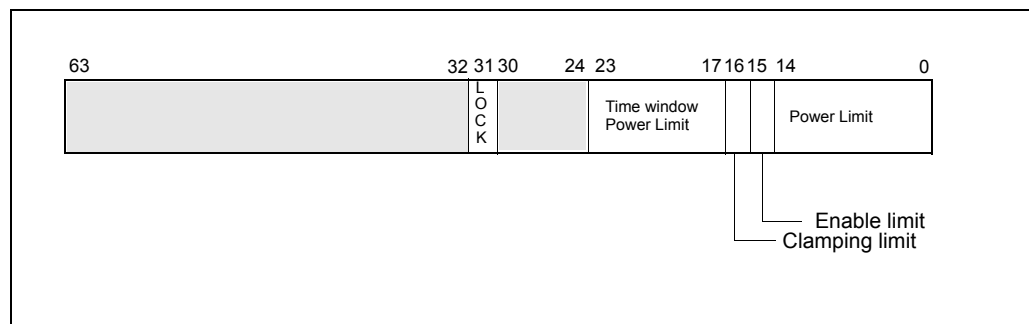
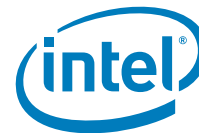


Figure 14-25 MSR_DRAM_POWER_LIMIT Register

MSR_DRAM_POWER_LIMIT allows a software agent to define power limitation for the DRAM domain. Power limitation is defined in terms of average power usage (Watts) over a time window specified in MSR_DRAM_POWER_LIMIT. A power limit can be specified along with a time window. A lock mechanism allow the software agent to enforce power limit settings. Once the lock bit is set, the power limit settings are static and un-modifiable until next RESET.

The bit fields of MSR_DRAM_POWER_LIMIT (Figure 14-25) are:



- **DRAM Power Limit #1**(bits 14:0): Sets the average power usage limit of the DRAM domain corresponding to time window # 1. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit #1**(bit 15): 0 = disabled; 1 = enabled.
- **Time Window for Power Limit** (bits 23:17): Indicates the length of time window over which the power limit The numeric value encoded by bits 23:17 is represented by the product of $2^Y * F$; where F is a single-digit decimal floating-point value between 1.0 and 1.3 with the fraction digit represented by bits 23:22, Y is an unsigned integer represented by bits 21:17. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Lock** (bit 31): If set, all write attempts to this MSR are ignored until next RESET.

MSR_DRAM_ENERGY_STATUS is a read-only MSR. It reports the actual energy use for the DRAM domain. This MSR is updated every ~1msec.

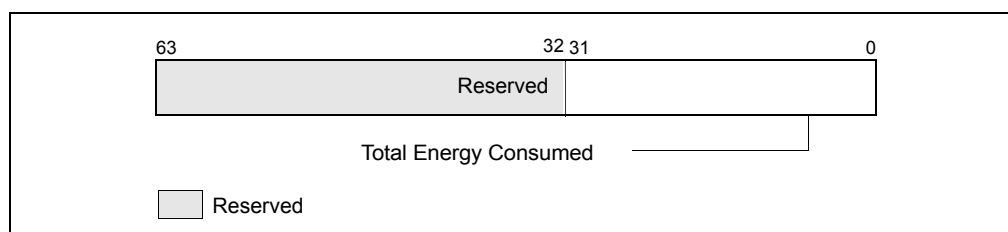


Figure 14-26 MSR_DRAM_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since that last time this register is cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_DRAM_POWER_INFO is a read-only MSR. It reports the DRAM power range information for RAPL usage. This MSR provides maximum/minimum values (derived from electrical specification), thermal specification power of the DRAM domain. It also provides the largest possible time window for software to program the RAPL interface.

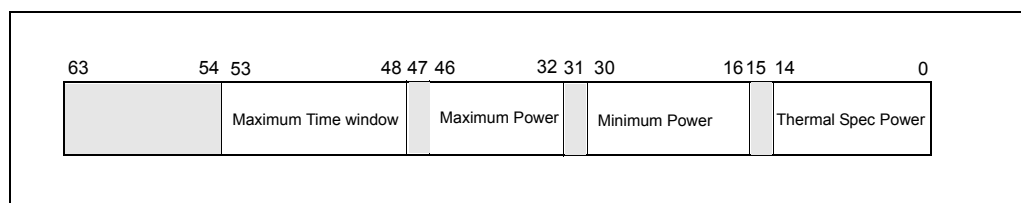


Figure 14-27 MSR_DRAM_POWER_INFO Register

- **Thermal Spec Power** (bits 14:0): The unsigned integer value is the equivalent of thermal specification power of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Minimum Power** (bits 30:16): The unsigned integer value is the equivalent of minimum power derived from electrical spec of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.



- **Maximum Power** (bits 46:32): The unsigned integer value is the equivalent of maximum power derived from the electrical spec of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Maximum Time Window** (bits 53:48): The unsigned integer value is the equivalent of largest acceptable value to program the time window of MSR_DRAM_POWER_LIMIT. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

...

11. Updates to Chapter 17, Volume 3B

Change bars show changes to Chapter 17 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

...

17.6.1 LBR Stack

Processors based on Intel microarchitecture code name Nehalem provide 16 pairs of MSR to record last branch record information. The layout of each MSR pair is shown in Table 17-6 and Table 17-7.

Table 17-6 IA32_LASTBRANCH_x_FROM_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/O	The linear address of the branch instruction itself, this is the "branch from" address.
SIGN_EXT	62:48	R/O	Signed extension of bit 47 of this register.
MISPRED	63	R/O	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

...

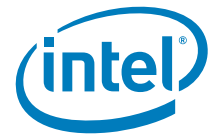
12. Updates to Chapter 18, Volume 3B

Change bars show changes to Chapter 18 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

...

18.8.7 Intel® Xeon® Processor E5 Family Performance Monitoring Facility

The Intel® Xeon® processor E5 Family (and Intel® Core™ i7-3930K Processor) are based on Intel microarchitecture code name Sandy Bridge. While the processor cores



share the same microarchitecture as those of the Intel® Xeon® Processor E3 Family and second generation Intel Core i7-2xxx, Intel Core i5-2xxx, Intel Core i3-2xxx processor series, the uncore subsystems are different. An overview of the uncore performance monitoring facilities of the Intel Xeon processor E5 family (and Intel Core i7-3930K processor) is described in Section 18.8.8.

Thus, the performance monitoring facilities in the processor core generally are the same as those described in Section 18.8 through Section 18.8.5. However, the MSR_OFFCORE_RSP_0/MSR_OFFCORE_RSP_1 Response Supplier Info field shown in Table 18-26 applies to Intel Core Processors with CPUID signature of DisplayFamily_DisplayModel encoding of 06_2AH; next generation Intel Xeon processor with CPUID signature of DisplayFamily_DisplayModel encoding of 06_2DH supports an additional field for remote DRAM controller shown in Table 18-29. Additionally, there are some small differences in the non-architectural performance monitoring events (see Table 19-4).

Table 18-29 MSR_OFFCORE_RSP_x Supplier Info Field Definition for Next Generation Intel Xeon Processor

Subtype	Bit Name	Offset	Description
Common	Any	16	(R/W). Catch all value for any response types.
Supplier Info	NO_SUPP	17	(R/W). No Supplier Information available
	LLC_HITM	18	(R/W). M-state initial lookup stat in L3.
	LLC_HITE	19	(R/W). E-state
	LLC_HITS	20	(R/W). S-state
	LLC_HITF	21	(R/W). F-state
	LOCAL	22	(R/W). Local DRAM Controller
	Remote	30:23	(R/W): Remote DRAM Controller (either all 0s or all 1s)

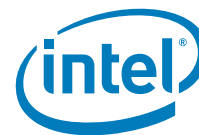
18.8.8 Intel® Xeon® Processor E5 Family Uncore Performance Monitoring Facility

The uncore subsystem in the Intel Xeon processor E5 family based on Intel microarchitecture Sandy Bridge has some similarities with those of the Intel Xeon processor E7 family based on Intel microarchitecture Sandy Bridge. Within the uncore subsystem, localized performance counter sets are provided at logic control unit scope. For example, each Cbox caching agent has a set of local performance counters, and the power controller unit (PCU) has its own local performance counters. Up to 8 C-Box units are supported in the uncore sub-system.

Table 18-30 summarizes the uncore PMU facilities providing MSR interfaces.

Table 18-30 Uncore PMU MSR Summary for Intel® Xeon® Processor E5 Family

Box	# of Boxes	Counters per Box	Counter Width	General Purpose	Global Enable	Sub-control MSRs
C-Box	8	4	44	Yes	per-box	None
PCU	1	4	48	Yes	per-box	Match/Mask
U-Box	1	2	44	Yes	uncore	None



...

13. Updates to Chapter 19, Volume 3B

Change bars show changes to Chapter 19 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

...

19.3 PERFORMANCE MONITORING EVENTS FOR 2ND GENERATION INTEL® CORE™ I7-2XXX, INTEL® CORE™ I5-2XXX, INTEL® CORE™ I3-2XXX PROCESSOR SERIES

Second generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series are based on the Intel microarchitecture code name Sandy Bridge. They support architectural performance-monitoring events listed in Table 19-1. Non-architectural performance-monitoring events in the processor core are listed in Table 19-3, Table 19-4, and Table 19-5. The events in Table 19-3 apply to processors with CPUID signature of DisplayFamily_DisplayModel encoding with the following values: 06_2AH and 06_2DH. The events in Table 19-4 apply to processors with CPUID signature 06_2AH. The events in Table 19-5 apply to processors with CPUID signature 06_2DH.

Table 19-3 Non-Architectural Performance Events In the Processor Core common to Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
03H	01H	LD_BLOCKS.DATA_UNKNOWN	blocked loads due to store buffer blocks with unknown data.	
03H	02H	LD_BLOCKS.STORE_FORWARD	loads blocked by overlapping with store buffer that cannot be forwarded .	
03H	08H	LD_BLOCKS.NO_SR	# of Split loads blocked due to resource not available.	
03H	10H	LD_BLOCKS.ALL_BLOCK	Number of cases where any load is blocked but has no DCU miss.	
05H	01H	MISALIGN_MEM_REF.LOADS	Speculative cache-line split load uops dispatched to L1D.	
05H	02H	MISALIGN_MEM_REF.STORES	Speculative cache-line split Store-address uops dispatched to L1D.	
07H	01H	LD_BLOCKS_PARTIAL_ADDRESS_ALIAS	False dependencies in MOB due to partial compare on address.	

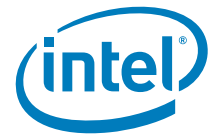


Table 19-3 Non-Architectural Performance Events In the Processor Core common to Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
07H	08H	LD_BLOCKS_PARTIAL.ALL_STA_BLOCK	The number of times that load operations are temporarily blocked because of older stores, with addresses that are not yet known. A load operation may incur more than one block of this type.	
08H	01H	DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK	Misses in all TLB levels that cause a page walk of any page size.	
08H	02H	DTLB_LOAD_MISSES.WALK_COMPLETED	Misses in all TLB levels that caused page walk completed of any size.	
08H	04H	DTLB_LOAD_MISSES.WALK_DURATION	Cycle PMH is busy with a walk.	
08H	10H	DTLB_LOAD_MISSES.STLB_HIT	Number of cache load STLB hits. No page walk.	
0DH	03H	INT_MISC.RECOVERY_CYCLES	Cycles waiting to recover after Machine Clears or JEClear. Set Cmask= 1.	Set Edge to count occurrences
0DH	40H	INT_MISC.RAT_STALL_CYCLES	Cycles RAT external stall is sent to IDQ for this thread.	
0EH	01H	UOPS_ISSUED.ANY	Increments each cycle the # of Uops issued by the RAT to RS. Set Cmask = 1, Inv = 1, Any= 1 to count stalled cycles of this core.	Set Cmask = 1, Inv = 1 to count stalled cycles
10H	01H	FP_COMP_OPS_EXE.X87	Counts number of X87 uops executed.	
10H	10H	FP_COMP_OPS_EXE.SSE_FP_PACKED_DOUBLE	Counts number of SSE* double precision FP packed uops executed.	
10H	20H	FP_COMP_OPS_EXE.SSE_FP_SCALAR_SINGLE	Counts number of SSE* single precision FP scalar uops executed.	
10H	40H	FP_COMP_OPS_EXE.SSE_PACKED_SINGLE	Counts number of SSE* single precision FP packed uops executed.	
10H	80H	FP_COMP_OPS_EXE.SSE_SCALAR_DOUBLE	Counts number of SSE* double precision FP scalar uops executed.	
11H	01H	SIMD_FP_256.PACKED_SINGLE	Counts 256-bit packed single-precision floating-point instructions	
11H	02H	SIMD_FP_256.PACKED_DOUBLE	Counts 256-bit packed double-precision floating-point instructions	



Table 19-3 Non-Architectural Performance Events In the Processor Core common to Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
14H	01H	ARITH.FPU_DIV_ACT IVE	Cycles that the divider is active, includes INT and FP. Set 'edge =1, cmask=1' to count the number of divides.	
17H	01H	INSTS_WRITTEN_TO _IQ.INSTS	Counts the number of instructions written into the IQ every cycle.	
24H	01H	L2_RQSTS.DEMAND_ DATA_RD_HIT	Demand Data Read requests that hit L2 cache	
24H	03H	L2_RQSTS.ALL_DEM AND_DATA_RD	Counts any demand and L1 HW prefetch data load requests to L2.	
24H	04H	L2_RQSTS.RFO_HITS	Counts the number of store RFO requests that hit the L2 cache.	
24H	08H	L2_RQSTS.RFO_MISS	Counts the number of store RFO requests that miss the L2 cache.	
24H	0CH	L2_RQSTS.ALL_RFO	Counts all L2 store RFO requests.	
24H	10H	L2_RQSTS.CODE_RD _HIT	Number of instruction fetches that hit the L2 cache.	
24H	20H	L2_RQSTS.CODE_RD _MISS	Number of instruction fetches that missed the L2 cache.	
24H	30H	L2_RQSTS.ALL_COD E_RD	Counts all L2 code requests.	
24H	40H	L2_RQSTS.PF_HIT	Requests from L2 Hardware prefetcher that hit L2.	
24H	80H	L2_RQSTS.PF_MISS	Requests from L2 Hardware prefetcher that missed L2.	
24H	C0H	L2_RQSTS.ALL_PF	Any requests from L2 Hardware prefetchers	
27H	01H	L2_STORE_LOCK_RQ STS.MISS	RFOs that miss cache lines	
27H	04H	L2_STORE_LOCK_RQ STS.HIT_E	RFOs that hit cache lines in E state	
27H	08H	L2_STORE_LOCK_RQ STS.HIT_M	RFOs that hit cache lines in M state	
27H	0FH	L2_STORE_LOCK_RQ STS.ALL	RFOs that access cache lines in any state	
28H	04H	L2_L1D_WB_RQSTS. HIT_E	Not rejected writebacks from L1D to L2 cache lines in E state.	
28H	08H	L2_L1D_WB_RQSTS. HIT_M	Not rejected writebacks from L1D to L2 cache lines in M state.	



Table 19-3 Non-Architectural Performance Events In the Processor Core common to Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
2EH	4FH	LONGEST_LAT_CACHE.REFERENCE	This event counts requests originating from the core that reference a cache line in the last level cache.	see Table 19-1
2EH	41H	LONGEST_LAT_CACHE.MISS	This event counts each cache miss condition for references to the last level cache.	see Table 19-1
3CH	00H	CPU_CLK_UNHALTED.THREAD_P	Counts the number of thread cycles while the thread is not in a halt state. The thread enters the halt state when it is running the HLT instruction. The core frequency may change from time to time due to power or thermal throttling.	see Table 19-1
3CH	01H	CPU_CLK_THREAD_UNHALTED.REF_XCLK	Increments at the frequency of XCLK (100 MHz) when not halted.	see Table 19-1
48H	01H	L1D_PEND_MISS.PENDING	Increments the number of outstanding L1D misses every cycle. Set Cmask = 1 and Edge =1 to count occurrences.	Counter 2 only; Set Cmask = 1 to count cycles.
49H	01H	DTLB_STORE_MISSES.MISS_CAUSES_A_WALK	Miss in all TLB levels causes a page walk of any page size (4K/2M/4M/1G).	
49H	02H	DTLB_STORE_MISSES.WALK_COMPLETED	Miss in all TLB levels causes a page walk that completes of any page size (4K/2M/4M/1G).	
49H	04H	DTLB_STORE_MISSES.WALK_DURATION	Cycles PMH is busy with this walk.	
49H	10H	DTLB_STORE_MISSES.STLB_HIT	Store operations that miss the first TLB level but hit the second and do not cause page walks	
4CH	01H	LOAD_HIT_PREF.SW_PREF	Not SW-prefetch load dispatches that hit fill buffer allocated for S/W prefetch.	
4CH	02H	LOAD_HIT_PREF.HW_PREF	Not SW-prefetch load dispatches that hit fill buffer allocated for H/W prefetch.	
4EH	02H	HW_PREF_REQ_DL1_MISS	Hardware Prefetch requests that miss the L1D cache. A request is being counted each time it access the cache & miss it, including if a block is applicable or if hit the Fill Buffer for example.	This accounts for both L1 streamer and IP-based (IPP) HW prefetchers.

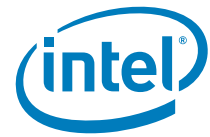


Table 19-3 Non-Architectural Performance Events In the Processor Core common to Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
51H	01H	L1D.REPLACEMENT	Counts the number of lines brought into the L1 data cache.	
51H	02H	L1D.ALLOCATED_IN_M	Counts the number of allocations of modified L1D cache lines.	
51H	04H	L1D.EVICTION	Counts the number of modified lines evicted from the L1 data cache due to replacement.	
51H	08H	L1D.ALL_M_REPLACEMENT	Cache lines in M state evicted out of L1D due to Snoop HitM or dirty line replacement	
59H	20H	PARTIAL_RATE_STALLS.FLAGS_MERGE_UOP	Increments the number of flags-merge uops in flight each cycle. Set Cmask = 1 to count cycles.	
59H	40H	PARTIAL_RATE_STALLS.SLOW_LEA_WINDOW	Cycles with at least one slow LEA uop allocated.	
59H	80H	PARTIAL_RATE_STALLS.MUL_SINGLE_UOP	Number of Multiply packed/scalar single precision uops allocated.	
5BH	0CH	RESOURCE_STALLS2.ALL_FL_EMPTY	Cycles stalled due to free list empty	
5BH	0FH	RESOURCE_STALLS2.ALL_PRF_CONTROL	Cycles stalled due to control structures full for physical registers	
5BH	40H	RESOURCE_STALLS2.BOB_FULL	Cycles Allocator is stalled due Branch Order Buffer.	
5BH	4FH	RESOURCE_STALLS2.OOO_RSRC	Cycles stalled due to out of order resources full	
5CH	01H	CPL_CYCLES.RING0	Unhalted core cycles when the thread is in ring 0	Use Edge to count transition
5CH	02H	CPL_CYCLES.RING123	Unhalted core cycles when the thread is not in ring 0	
5EH	01H	RS_EVENTS.EMPTY_CYCLES	Cycles the RS is empty for the thread.	
60H	01H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_DATA_RD	Offcore outstanding Demand Data Read transactions in SQ to uncore. Set Cmask=1 to count cycles.	
60H	04H	OFFCORE_REQUESTS_OUTSTANDING.DEMAND_RFO	Offcore outstanding RFO store transactions in SQ to uncore. Set Cmask=1 to count cycles.	
60H	08H	OFFCORE_REQUESTS_OUTSTANDING.ALL_DATA_RD	Offcore outstanding cacheable data read transactions in SQ to uncore. Set Cmask=1 to count cycles.	

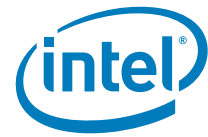


Table 19-3 Non-Architectural Performance Events In the Processor Core common to Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
63H	01H	LOCK_CYCLES.SPLIT_LOCK_UC_LOCK_DURATION	Cycles in which the L1D and L2 are locked, due to a UC lock or split lock.	
63H	02H	LOCK_CYCLES.CACHE_LOCK_DURATION	Cycles in which the L1D is locked.	
79H	02H	IDQ.EMPTY	Counts cycles the IDQ is empty.	
79H	04H	IDQ.MITE_UOPS	Increment each cycle # of uops delivered to IDQ from MITE path. Set Cmask = 1 to count cycles.	Can combine Umask 04H and 20H
79H	08H	IDQ.DSB_UOPS	Increment each cycle. # of uops delivered to IDQ from DSB path. Set Cmask = 1 to count cycles.	Can combine Umask 08H and 10H
79H	10H	IDQ.MS_DSB_UOPS	Increment each cycle # of uops delivered to IDQ when MS busy by DSB. Set Cmask = 1 to count cycles MS is busy. Set Cmask=1 and Edge =1 to count MS activations.	Can combine Umask 08H and 10H
79H	20H	IDQ.MS_MITE_UOPS	Increment each cycle # of uops delivered to IDQ when MS is busy by MITE. Set Cmask = 1 to count cycles.	Can combine Umask 04H and 20H
79H	30H	IDQ.MS_UOPS	Increment each cycle # of uops delivered to IDQ from MS by either DSB or MITE. Set Cmask = 1 to count cycles.	Can combine Umask 04H, 08H and 30H
80H	02H	ICACHE.MISSES	Number of Instruction Cache, Streaming Buffer and Victim Cache Misses. Includes UC accesses.	
85H	01H	ITLB_MISSES.MISS_CAUSES_A_WALK	Misses in all ITLB levels that cause page walks	
85H	02H	ITLB_MISSES.WALK_COMPLETED	Misses in all ITLB levels that cause completed page walks	
85H	04H	ITLB_MISSES.WALK_DURATION	Cycle PMH is busy with a walk.	
85H	10H	ITLB_MISSES.STLB_HIT	Number of cache load STLB hits. No page walk.	
87H	01H	ILD_STALL.LCP	Stalls caused by changing prefix length of the instruction.	
87H	04H	ILD_STALL.IQ_FULL	Stall cycles due to IQ is full.	
88H	01H	BR_INST_EXEC.COND	Qualify conditional near branch instructions executed, but not necessarily retired.	Must combine with umask 40H, 80H

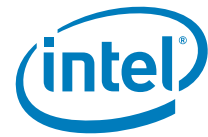


Table 19-3 Non-Architectural Performance Events In the Processor Core common to Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
88H	02H	BR_INST_EXEC.DIRECT_JMP	Qualify all unconditional near branch instructions excluding calls and indirect branches.	Must combine with umask 80H
88H	04H	BR_INST_EXEC.INDIRECT_JMP_NON_CALL_RET	Qualify executed indirect near branch instructions that are not calls nor returns.	Must combine with umask 80H
88H	08H	BR_INST_EXEC.RETURN_NEAR	Qualify indirect near branches that have a return mnemonic.	Must combine with umask 80H
88H	10H	BR_INST_EXEC.DIRECT_NEAR_CALL	Qualify unconditional near call branch instructions, excluding non call branch, executed.	Must combine with umask 80H
88H	20H	BR_INST_EXEC.INDIRECT_NEAR_CALL	Qualify indirect near calls, including both register and memory indirect, executed.	Must combine with umask 80H
88H	40H	BR_INST_EXEC.NON_TAKEN	Qualify non-taken near branches executed.	Applicable to umask 01H only
88H	80H	BR_INST_EXEC.TAKEN	Qualify taken near branches executed. Must combine with 01H,02H, 04H, 08H, 10H, 20H	
88H	FFH	BR_INST_EXEC.ALL_BRANCHES	Counts all near executed branches (not necessarily retired).	
89H	01H	BR_MISP_EXEC.COND	Qualify conditional near branch instructions mispredicted.	Must combine with umask 40H, 80H
89H	04H	BR_MISP_EXEC.INDIRECT_JMP_NON_CALL_RET	Qualify mispredicted indirect near branch instructions that are not calls nor returns.	Must combine with umask 80H
89H	08H	BR_MISP_EXEC.RETURN_NEAR	Qualify mispredicted indirect near branches that have a return mnemonic.	Must combine with umask 80H
89H	10H	BR_MISP_EXEC.DIRECT_NEAR_CALL	Qualify mispredicted unconditional near call branch instructions, excluding non call branch, executed.	Must combine with umask 80H
89H	20H	BR_MISP_EXEC.INDIRECT_NEAR_CALL	Qualify mispredicted indirect near calls, including both register and memory indirect, executed.	Must combine with umask 80H
89H	40H	BR_MISP_EXEC.NON_TAKEN	Qualify mispredicted non-taken near branches executed,.	Applicable to umask 01H only
89H	80H	BR_MISP_EXEC.TAKEN	Qualify mispredicted taken near branches executed. Must combine with 01H,02H, 04H, 08H, 10H, 20H	
89H	FFH	BR_MISP_EXEC.ALL_BRANCHES	Counts all near executed branches (not necessarily retired).	

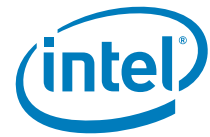


Table 19-3 Non-Architectural Performance Events In the Processor Core common to Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
9CH	01H	IDQ_UOPS_NOT_DELIVERED.CORE	Count number of non-delivered uops to RAT per thread.	Use Cmask to qualify uop b/w
A1H	01H	UOPS_DISPATCHED_PORT.PORT_0	Cycles which a Uop is dispatched on port 0.	
A1H	02H	UOPS_DISPATCHED_PORT.PORT_1	Cycles which a Uop is dispatched on port 1.	
A1H	04H	UOPS_DISPATCHED_PORT.PORT_2_LD	Cycles which a load uop is dispatched on port 2.	
A1H	08H	UOPS_DISPATCHED_PORT.PORT_2_STA	Cycles which a store address uop is dispatched on port 2.	
A1H	0CH	UOPS_DISPATCHED_PORT.PORT_2	Cycles which a Uop is dispatched on port 2.	
A1H	10H	UOPS_DISPATCHED_PORT.PORT_3_LD	Cycles which a load uop is dispatched on port 3.	
A1H	20H	UOPS_DISPATCHED_PORT.PORT_3_STA	Cycles which a store address uop is dispatched on port 3.	
A1H	30H	UOPS_DISPATCHED_PORT.PORT_3	Cycles which a Uop is dispatched on port 3.	
A1H	40H	UOPS_DISPATCHED_PORT.PORT_4	Cycles which a Uop is dispatched on port 4.	
A1H	80H	UOPS_DISPATCHED_PORT.PORT_5	Cycles which a Uop is dispatched on port 5.	
A2H	01H	RESOURCE_STALLS.ANY	Cycles Allocation is stalled due to Resource Related reason.	
A2H	02H	RESOURCE_STALLS.LB	Counts the cycles of stall due to lack of load buffers.	
A2H	04H	RESOURCE_STALLS.RS	Cycles stalled due to no eligible RS entry available.	
A2H	08H	RESOURCE_STALLS.SB	Cycles stalled due to no store buffers available. (not including draining form sync).	
A2H	10H	RESOURCE_STALLS.ROB	Cycles stalled due to re-order buffer full.	
A2H	20H	RESOURCE_STALLS.FCSW	Cycles stalled due to writing the FPU control word.	
A2H	40H	RESOURCE_STALLS.MXCSR	Cycles stalled due to the MXCSR register rename occurring to close to a previous MXCSR rename.	
A2H	80H	RESOURCE_STALLS.OTHER	Cycles stalled while execution was stalled due to other resource issues.	
ABH	01H	DSB2MITE_SWITCHES.COUNT	Number of DSB to MITE switches.	

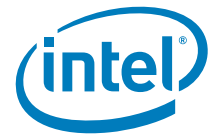


Table 19-3 Non-Architectural Performance Events In the Processor Core common to Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
ABH	02H	DSB2MITE_SWITCHES.PENALTY_CYCLES	Cycles DSB to MITE switches caused delay.	
ACH	02H	DSB_FILL.OTHER_CANCEL	Cases of cancelling valid DSB fill not because of exceeding way limit	
ACH	08H	DSB_FILL.EXCEEDED_DSB_LINES	DSB Fill encountered > 3 DSB lines.	
ACH	0AH	DSB_FILL.ALL_CANCEL	Cases of cancelling valid Decode Stream Buffer (DSB) fill not because of exceeding way limit	
AEH	01H	ITLB.ITLB_FLUSH	Counts the number of ITLB flushes, includes 4k/2M/4M pages.	
BOH	01H	OFFCORE_REQUESTS.DEMAND_DATA_RD	Demand data read requests sent to uncore.	
BOH	04H	OFFCORE_REQUESTS.DEMAND_RFO	Demand RFO read requests sent to uncore., including regular RFOs, locks, ItoM	
BOH	08H	OFFCORE_REQUESTS.ALL_DATA_RD	Data read requests sent to uncore (demand and prefetch).	
B1H	01H	UOPS_DISPATCHED.THREAD	Counts total number of uops to be dispatched per-thread each cycle. Set Cmask = 1, INV =1 to count stall cycles.	
B1H	02H	UOPS_DISPATCHED.CORE	Counts total number of uops to be dispatched per-core each cycle.	Do not need to set ANY
B2H	01H	OFFCORE_REQUESTS.BUFFER_SQ_FULL	Offcore requests buffer cannot take more entries for this thread core.	
B6H	01H	AGU_BYPASS_CANCEL.COUNT	Counts executed load operations with all the following traits: 1. addressing of the format [base + offset], 2. the offset is between 1 and 2047, 3. the address specified in the base register is in one page and the address [base+offset] is in another page.	
B7H	01H	OFF_CORE_RESPONSE_0	see Section 18.8.5, "Off-core Response Performance Monitoring"; PMCO only.	Requires programming MSR 01A6H
BBH	01H	OFF_CORE_RESPONSE_1	See Section 18.8.5, "Off-core Response Performance Monitoring". PMC3 only.	Requires programming MSR 01A7H
BDH	01H	TLB_FLUSH.DTLB_THREAD	DTLB flush attempts of the thread-specific entries	

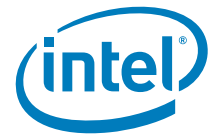


Table 19-3 Non-Architectural Performance Events In the Processor Core common to Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
BDH	20H	TLB_FLUSH.STLB_ANY	Count number of STLB flush attempts	
BFH	05H	L1D_BLOCKS.BANK_CONFLICT_CYCLES	Cycles when dispatched loads are cancelled due to L1D bank conflicts with other load ports	cmask=1
COH	00H	INST_RETIRED.ANY_P	Number of instructions at retirement	See Table 19-1
COH	01H	INST_RETIRED.PREC_DIST	Precise instruction retired event with HW to reduce effect of PEBS shadow in IP distribution	PMC1 only; Must quiesce other PMCs.
C1H	02H	OTHER_ASSISTS.ITLB_MISS_RETIRED	Instructions that experienced an ITLB miss.	
C1H	08H	OTHER_ASSISTS.AVX_STORE	Number of assists associated with 256-bit AVX store operations.	
C1H	10H	OTHER_ASSISTS.AVX_TO_SSE	Number of transitions from AVX-256 to legacy SSE when penalty applicable.	
C1H	20H	OTHER_ASSISTS.SSE_TO_AVX	Number of transitions from SSE to AVX-256 when penalty applicable.	
C2H	01H	UOPS_RETIRED.ALL	Counts the number of micro-ops retired. Use cmask=1 and invert to count active cycles or stalled cycles.	Supports PEBS
C2H	02H	UOPS_RETIRED.RETIRE_SLOTS	Counts the number of retirement slots used each cycle.	
C3H	02H	MACHINE_CLEAR.MEMORY_ORDERING	Counts the number of machine clears due to memory order conflicts.	
C3H	04H	MACHINE_CLEAR.SMC	Counts the number of times that a program writes to a code section.	
C3H	20H	MACHINE_CLEAR.MASKMOV	Counts the number of executed AVX masked load operations that refer to an illegal address range with the mask bits set to 0.	
C4H	00H	BR_INST_RETIRED.ALL_BRANCHES	Branch instructions at retirement	See Table 19-1
C4H	01H	BR_INST_RETIRED.CONDITIONAL	Counts the number of conditional branch instructions retired.	Supports PEBS
C4H	02H	BR_INST_RETIRED.NEAR_CALL	Direct and indirect near call instructions retired.	
C4H	04H	BR_INST_RETIRED.ALL_BRANCHES	Counts the number of branch instructions retired.	



Table 19-3 Non-Architectural Performance Events In the Processor Core common to Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C4H	08H	BR_INST_RETIRED.NEAR_RETURN	Counts the number of near return instructions retired.	
C4H	10H	BR_INST_RETIRED.NOT_TAKEN	Counts the number of not taken branch instructions retired.	
C4H	20H	BR_INST_RETIRED.NEAR_TAKEN	Number of near taken branches retired.	
C4H	40H	BR_INST_RETIRED.FAR_BRANCH	Number of far branches retired.	
C5H	00H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted branch instructions at retirement	See Table 19-1
C5H	01H	BR_MISP_RETIRED.CONDITIONAL	Mispredicted conditional branch instructions retired.	Supports PEBS
C5H	02H	BR_MISP_RETIRED.NEAR_CALL	Direct and indirect mispredicted near call instructions retired.	
C5H	04H	BR_MISP_RETIRED.ALL_BRANCHES	Mispredicted macro branch instructions retired.	
C5H	10H	BR_MISP_RETIRED.NOT_TAKEN	Mispredicted not taken branch instructions retired.	
C5H	20H	BR_MISP_RETIRED.TAKEN	Mispredicted taken branch instructions retired.	
CAH	02H	FP_ASSIST.X87_OUTPUT	Number of X87 assists due to output value.	
CAH	04H	FP_ASSIST.X87_INPUT	Number of X87 assists due to input value.	
CAH	08H	FP_ASSIST.SIMD_OUTPUT	Number of SIMD FP assists due to Output values	
CAH	10H	FP_ASSIST.SIMD_INPUT	Number of SIMD FP assists due to input values	
CAH	1EH	FP_ASSIST.ANY	Cycles with any input/output SSE* or FP assists	
CCH	20H	ROB_MISC_EVENTS.LBR_INSERTS	Count cases of saving new LBR records by hardware.	
CDH	01H	MEM_TRANS_RETIRED.LOAD_LATENCY	Sample loads with specified latency threshold. PMC3 only.	Specify threshold in MSR 0x3F6
CDH	02H	MEM_TRANS_RETIRED.PRECISE_STORE	Sample stores and collect precise store operation via PEBS record. PMC3 only.	See Section 18.8.4.3
DOH	01H	MEM_UOP_RETIRED.LOADS	Qualify retired memory uops that are loads. Combine with umask 10H, 20H, 40H, 80H.	Supports PEBS

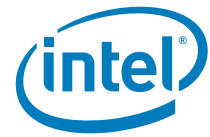


Table 19-3 Non-Architectural Performance Events In the Processor Core common to Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
DOH	02H	MEM_UOP_RETIRED.STORES	Qualify retired memory uops that are stores. Combine with umask 10H, 20H, 40H, 80H.	
DOH	10H	MEM_UOP_RETIRED.STLB_MISS	Qualify retired memory uops with STLB miss. Must combine with umask 01H, 02H, to produce counts.	
DOH	20H	MEM_UOP_RETIRED.LOCK	Qualify retired memory uops with lock. Must combine with umask 01H, 02H, to produce counts.	
DOH	40H	MEM_UOP_RETIRED.SPLIT	Qualify retired memory uops with line split. Must combine with umask 01H, 02H, to produce counts.	
DOH	80H	MEM_UOP_RETIRED.ALL	Qualify any retired memory uops. Must combine with umask 01H, 02H, to produce counts.	
D1H	01H	MEM_LOAD_UOPS_RETIRED.L1_HIT	Retired load uops with L1 cache hits as data sources.	Supports PEBS
D1H	02H	MEM_LOAD_UOPS_RETIRED.L2_HIT	Retired load uops with L2 cache hits as data sources.	
D1H	40H	MEM_LOAD_UOPS_RETIRED.HIT_LFB	Retired load uops which data sources were load uops missed L1 but hit FB due to preceding miss to the same cache line with data not ready.	
D2H	01H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_MISS	Retired load uops which data sources were LLC hit and cross-core snoop missed in on-pkg core cache.	Supports PEBS
D2H	02H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT	Retired load uops which data sources were LLC and cross-core snoop hits in on-pkg core cache.	
D2H	04H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM	Retired load uops which data sources were HitM responses from shared LLC.	
D2H	08H	MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE	Retired load uops which data sources were hits in LLC without snoops required.	
D4H	02H	MEM_LOAD_UOPS_MISSED.LLC_MISS	Retired load uops with unknown information as data source in cache serviced the load.	Supports PEBS.
FOH	01H	L2_TRANS.DEMAND_DATA_RD	Demand Data Read requests that access L2 cache	
FOH	02H	L2_TRANS.RFO	RFO requests that access L2 cache	

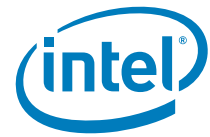


Table 19-3 Non-Architectural Performance Events In the Processor Core common to Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx Processor Series and Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
F0H	04H	L2_TRANS.CODE_RD	L2 cache accesses when fetching instructions	
F0H	08H	L2_TRANS.ALL_PF	L2 or LLC HW prefetches that access L2 cache	including rejects.
F0H	10H	L2_TRANS.L1D_WB	L1D writebacks that access L2 cache	
F0H	20H	L2_TRANS.L2_FILL	L2 fill requests that access L2 cache	
F0H	40H	L2_TRANS.L2_WB	L2 writebacks that access L2 cache	
F0H	80H	L2_TRANS.ALL_REQ_UESTS	Transactions accessing L2 pipe	
F1H	01H	L2_LINES_IN.I	L2 cache lines in I state filling L2	Counting does not cover rejects.
F1H	02H	L2_LINES_IN.S	L2 cache lines in S state filling L2	Counting does not cover rejects.
F1H	04H	L2_LINES_IN.E	L2 cache lines in E state filling L2	Counting does not cover rejects.
F1H	07H	L2_LINES_IN.ALL	L2 cache lines filling L2	Counting does not cover rejects.
F2H	01H	L2_LINES_OUT.DEMAND_CLEAN	Clean L2 cache lines evicted by demand	
F2H	02H	L2_LINES_OUT.DEMAND_DIRTY	Dirty L2 cache lines evicted by demand	
F2H	04H	L2_LINES_OUT.PF_CLEAN	Clean L2 cache lines evicted by L2 prefetch	
F2H	08H	L2_LINES_OUT.PF_DIRTY	Dirty L2 cache lines evicted by L2 prefetch	
F2H	0AH	L2_LINES_OUT.DIRTY_ALL	Dirty L2 cache lines filling the L2	Counting does not cover rejects.
F4H	10H	SQ_MISC.SPLIT_LOCK	Split locks in SQ	

...

Non-architecture performance monitoring events in the processor core that are applicable only to Intel Xeon processor E5 family (and Intel Core i7-3930 processor) based on Intel microarchitecture Sandy Bridge, with CPUID signature of DisplayFamily_DisplayModel 06_2DH, are listed in Table 19-5.

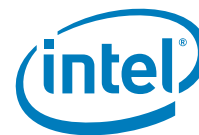


Table 19-5 Non-Architectural Performance Events Applicable only to the Processor Core of Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
B7H/ BBH	01H	OFF_CORE_RESPONS E_N	Sub-events of OFF_CORE_RESPONSE_N (suffix N = 0, 1) programmed using MSR 01A6H/ 01A7H with values shown in the comment column.	
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_MISS.ANY_RE SPONSE_N		0x3FFFC0000 4
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_MISS.LOCAL_D RAM_N		0x600400004
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_MISS.REMOTE _DRAM_N		0x67F800004
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_MISS.REMOTE _HIT_FWD_N		0x87F800004
		OFFCORE_RESPONSE.DEMAND_CODE_RD.LLC_MISS.REMOTE _HITM_N		0x107FC0000 4
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.ANY_DR AM_N		0x67FC00001
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.ANY_RE SPONSE_N		0x3F803C000 1
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.LOCAL_D RAM_N		0x600400001
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.REMOTE _DRAM_N		0x67F800001
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.REMOTE _HIT_FWD_N		0x87F800001
		OFFCORE_RESPONSE.DEMAND_DATA_RD.LLC_MISS.REMOTE _HITM_N		0x107FC0000 1
		OFFCORE_RESPONSE.PF_L2_CODE_RD.LLC_MISS.ANY_RESP ONSE_N		0x3F803C004 0
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.ANY_DRAM _N		0x67FC00010
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.ANY_RESP ONSE_N		0x3F803C001 0
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.LOCAL_DR AM_N		0x600400010
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.REMOTE_D RAM_N		0x67F800010
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.REMOTE_HI T_FWD_N		0x87F800010
		OFFCORE_RESPONSE.PF_L2_DATA_RD.LLC_MISS.REMOTE_HI TM_N		0x107FC0001 0



Table 19-5 Non-Architectural Performance Events Applicable only to the Processor Core of Intel® Xeon® Processor E5 Family

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
		OFFCORE_RESPONSE.PF_LLC_CODE_RD.LLC_MISS.ANY_RESPONSE_N		0x3FFFC00200
		OFFCORE_RESPONSE.PF_LLC_DATA_RD.LLC_MISS.ANY_RESPONSE_N		0x3FFFC00080

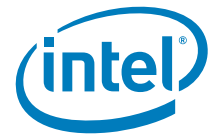
...

Table 19-7 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
...				
COH	00H	INST_RETIRED.ANY_P	See Table 19-1 Notes: INST_RETIRED.ANY is counted by a designated fixed counter. INST_RETIRED.ANY_P is counted by a programmable counter and is an architectural performance event. Event is supported if CPUID.A.EBX[1] = 0.	Counting: Faulting executions of GETSEC/VM entry/VM Exit/MWait will not count as retired instructions.
...				

...

Table 19-9 Non-Architectural Performance Events In the Processor Core for Processors



Based on Intel® Microarchitecture Code Name Westmere

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
...				
2EH	41H	L3_LAT_CACHE.MISS	Counts uncore Last Level Cache misses. Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.	see Table 19-1
2EH	4FH	L3_LAT_CACHE.REFERENCE	Counts uncore Last Level Cache references. Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.	see Table 19-1
...				
COH	00H	INST_RETIRED.ANY_P	See Table 19-1 Notes: INST_RETIRED.ANY is counted by a designated fixed counter. INST_RETIRED.ANY_P is counted by a programmable counter and is an architectural performance event. Event is supported if CPUID.A.EBX[1] = 0.	Counting: Faulting executions of GETSEC/VM entry/VM Exit/MWait will not count as retired instructions.
...				

...

14. Updates to Chapter 25, Volume 3C

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

...

25.3 OTHER CAUSES OF VM EXITS

In addition to VM exits caused by instruction execution, the following events can cause VM exits:

- **Exceptions.** Exceptions (faults, traps, and aborts) cause VM exits based on the exception bitmap (see Section 24.6.3). If an exception occurs, its vector (in the range 0–31) is used to select a bit in the exception bitmap. If the bit is 1, a VM exit occurs; if the bit is 0, the exception is delivered normally through the guest IDT. This use of the exception bitmap applies also to exceptions generated by the instructions INT3, INTO, BOUND, and UD2.

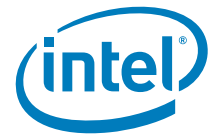


Page faults (exceptions with vector 14) are specially treated. When a page fault occurs, a logical processor consults (1) bit 14 of the exception bitmap; (2) the error code produced with the page fault [PFEC]; (3) the page-fault error-code mask field [PFEC_MASK]; and (4) the page-fault error-code match field [PFEC_MATCH]. It checks if $PFEC \& PFEC_MASK = PFEC_MATCH$. If there is equality, the specification of bit 14 in the exception bitmap is followed (for example, a VM exit occurs if that bit is set). If there is inequality, the meaning of that bit is reversed (for example, a VM exit occurs if that bit is clear).

Thus, if software desires VM exits on all page faults, it can set bit 14 in the exception bitmap to 1 and set the page-fault error-code mask and match fields each to 00000000H. If software desires VM exits on no page faults, it can set bit 14 in the exception bitmap to 1, the page-fault error-code mask field to 00000000H, and the page-fault error-code match field to FFFFFFFFH.

- **Triple fault.** A VM exit occurs if the logical processor encounters an exception while attempting to call the double-fault handler and that exception itself does not cause a VM exit due to the exception bitmap. This applies to the case in which the double-fault exception was generated within VMX non-root operation, the case in which the double-fault exception was generated during event injection by VM entry, and to the case in which VM entry is injecting a double-fault exception.
- **External interrupts.** An external interrupt causes a VM exit if the “external-interrupt exiting” VM-execution control is 1. Otherwise, the interrupt is delivered normally through the IDT. (If a logical processor is in the shutdown state or the wait-for-SIPI state, external interrupts are blocked. The interrupt is not delivered through the IDT and no VM exit occurs.)
- **Non-maskable interrupts (NMIs).** An NMI causes a VM exit if the “NMI exiting” VM-execution control is 1. Otherwise, it is delivered using descriptor 2 of the IDT. (If a logical processor is in the wait-for-SIPI state, NMIs are blocked. The NMI is not delivered through the IDT and no VM exit occurs.)
- **INIT signals.** INIT signals cause VM exits. A logical processor performs none of the operations normally associated with these events. Such exits do not modify register state or clear pending events as they would outside of VMX operation. (If a logical processor is in the wait-for-SIPI state, INIT signals are blocked. They do not cause VM exits in this case.)
- **Start-up IPIs (SIPIs). SIPIs cause VM exits.** If a logical processor is not in the wait-for-SIPI activity state when a SIPI arrives, no VM exit occurs and the SIPI is discarded. VM exits due to SIPIs do not perform any of the normal operations associated with those events: they do not modify register state as they would outside of VMX operation. (If a logical processor is not in the wait-for-SIPI state, SIPIs are blocked. They do not cause VM exits in this case.)
- **Task switches.** Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. See Section 25.6.2.
- **System-management interrupts (SMIs).** If the logical processor is using the dual-monitor treatment of SMIs and system-management mode (SMM), SMIs cause SMM VM exits. See Section 33.15.2.¹
- **VMX-preemption timer.** A VM exit occurs when the timer counts down to zero. See Section 25.7.1 for details of operation of the VMX-preemption timer.

1. Under the dual-monitor treatment of SMIs and SMM, SMIs also cause SMM VM exits if they occur in VMX root operation outside SMM. If the processor is using the default treatment of SMIs and SMM, SMIs are delivered as described in Section 33.14.1.



Debug-trap exceptions and higher priority events take priority over VM exits caused by the VMX-preemption timer. VM exits caused by the VMX-preemption timer take priority over VM exits caused by the “NMI-window exiting” VM-execution control and lower priority events.

These VM exits wake a logical processor from the same inactive states as would a non-maskable interrupt. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

...

25.7.1 VMX-Preemption Timer

If the last VM entry was performed with the 1-setting of “activate VMX-preemption timer” VM-execution control, the **VMX-preemption timer** counts down (from the value loaded by VM entry; see Section 26.6.4) in VMX non-root operation. When the timer counts down to zero, it stops counting down and a VM exit occurs (see Section 25.3).

The VMX-preemption timer counts down at rate proportional to that of the timestamp counter (TSC). Specifically, the timer counts down by 1 every time bit X in the TSC changes due to a TSC increment. The value of X is in the range 0–31 and can be determined by consulting the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

The VMX-preemption timer operates in the C-states C0, C1, and C2; it also operates in the shutdown and wait-for-SIPI states. If the timer counts down to zero in any state other than the wait-for SIPI state, the logical processor transitions to the C0 C-state and causes a VM exit; the timer does not cause a VM exit if it counts down to zero in the wait-for-SIPI state. The timer is not decremented in C-states deeper than C2.

...

15. Updates to Chapter 26, Volume 3C

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

...

26.2.1.1 VM-Execution Control Fields

VM entries perform the following checks on the VM-execution control fields:¹

- Reserved bits in the pin-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.1).
- Reserved bits in the primary processor-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.2).
- If the “activate secondary controls” primary processor-based VM-execution control is 1, reserved bits in the secondary processor-based VM-execution controls must be

1. If the “activate secondary controls” primary processor-based VM-execution control is 0, VM entry operates as if each secondary processor-based VM-execution control were 0.



cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.3.3).

If the “activate secondary controls” primary processor-based VM-execution control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary processor-based VM-execution controls. The logical processor operates as if all the secondary processor-based VM-execution controls were 0.

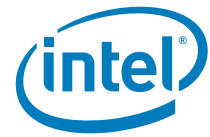
- The CR3-target count must not be greater than 4. Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC to determine the number of values supported (see Appendix A.6).
- If the “use I/O bitmaps” VM-execution control is 1, bits 11:0 of each I/O-bitmap address must be 0. Neither address should set any bits beyond the processor’s physical-address width.^{1,2}
- If the “use MSR bitmaps” VM-execution control is 1, bits 11:0 of the MSR-bitmap address must be 0. The address should not set any bits beyond the processor’s physical-address width.³
- If the “use TPR shadow” VM-execution control is 1, the virtual-APIC address must satisfy the following checks:
 - Bits 11:0 of the address must be 0.
 - The address should not set any bits beyond the processor’s physical-address width.⁴

If all of the above checks are satisfied and the “use TPR shadow” VM-execution control is 1, bytes 81H-83H on the virtual-APIC page (see Section 24.6.8) may be cleared (behavior may be implementation-specific).

The clearing of these bytes may occur even if the VM entry fails. This is true either if the failure causes control to pass to the instruction following the VM-entry instruction or if it causes processor state to be loaded from the host-state area of the VMCS.

- If the “use TPR shadow” VM-execution control is 1, bits 31:4 of the TPR threshold VM-execution control field must be 0.
- The following check is performed if the “use TPR shadow” VM-execution control is 1 and the “virtualize APIC accesses” VM-execution control is 0: the value of bits 3:0 of the TPR threshold VM-execution control field should not be greater than the value of bits 7:4 in byte 80H on the virtual-APIC page (see Section 24.6.8).
- If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” VM-execution control must be 0.
- If the “virtual NMIs” VM-execution control is 0, the “NMI-window exiting” VM-execution control must be 0.

-
1. 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.
 2. If IA32_VMX_BASIC[48] is read as 1, these addresses must not set any bits in the range 63:32; see Appendix A.1.
 3. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.
 4. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.



- If the “virtualize APIC-accesses” VM-execution control is 1, the APIC-access address must satisfy the following checks:
 - Bits 11:0 of the address must be 0.
 - The address should not set any bits beyond the processor’s physical-address width.¹

...

16. Updates to Chapter 27, Volume 3C

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

...

27.2.4 Information for VM Exits Due to Instruction Execution

Section 24.9.4 defined fields containing information for VM exits that occur due to instruction execution. (The VM-exit instruction length is also used for VM exits that occur during the delivery of a software interrupt or software exception.) The following items detail their use.

- **VM-exit instruction length.** This field is used in the following cases:
 - For fault-like VM exits due to attempts to execute one of the following instructions that cause VM exits unconditionally (see Section 25.1.2) or based on the settings of VM-execution controls (see Section 25.1.3): CLTS, CPUID, GETSEC, HLT, IN, INS, INVD, INVEPT, INVLPG, INVPCID, INVVPID, LGDT, LIDT, LLDT, LMSW, LTR, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, RDMSR, RDPMSR, RDRAND, RDTSC, RDTSCP, RSM, SGDT, SIDT, SLDT, STR, VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON, WBINVD, WRMSR, and XSETBV.²
 - For VM exits due to software exceptions (those generated by executions of INT3 or INTO).
 - For VM exits due to faults encountered during delivery of a software interrupt, privileged software exception, or software exception.
 - For VM exits due to attempts to effect a task switch via instruction execution. These are VM exits that produce an exit reason indicating task switch and either of the following:
 - An exit qualification indicating execution of CALL, IRET, or JMP instruction.
 - An exit qualification indicating a task gate in the IDT and an IDT-vectoring information field indicating that the task gate was encountered during

-
1. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.
 2. This item applies only to fault-like VM exits. It does not apply to trap-like VM exits following executions of the MOV to CR8 instruction when the “use TPR shadow” VM-execution control is 1 or to those following executions of the WRMSR instruction when the “virtualize x2APIC mode” VM-execution control is 1.



delivery of a software interrupt, privileged software exception, or software exception.

- For APIC-access VM exits resulting from linear accesses (see Section 25.2.1) and encountered during delivery of a software interrupt, privileged software exception, or software exception.¹
- For VM exits due executions of VMFUNC that fail because one of the following is true:
 - EAX indicates a VM function that is not enabled (the bit at position EAX is 0 in the VM-function controls; see Section 25.7.4.2).
 - EAX = 0 and either ECX ≥ 512 or the value of ECX selects an invalid tentative EPTP value (see Section 25.7.4.3).

In all the above cases, this field receives the length in bytes (1–15) of the instruction (including any instruction prefixes) whose execution led to the VM exit (see the next paragraph for one exception).

The cases of VM exits encountered during delivery of a software interrupt, privileged software exception, or software exception include those encountered during delivery of events injected as part of VM entry (see Section 26.5.1.2). If the original event was injected as part of VM entry, this field receives the value of the VM-entry instruction length.

All VM exits other than those listed in the above items leave this field undefined.

...

17. Update to Volume 3C

Chapter 29 and chapter 33 were swapped in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C: System Programming Guide, Part 3*.

18. Updates to Chapter 33, Volume 3C

Change bars show changes to Chapter 33 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C: System Programming Guide, Part 3*.

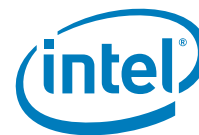
...

33.4.2 SMRAM Caching

An IA-32 processor does not automatically write back and invalidate its caches before entering SMM or before exiting SMM. Because of this behavior, care must be taken in the placement of the SMRAM in system memory and in the caching of the SMRAM to prevent cache incoherence when switching back and forth between SMM and protected mode operation. Either of the following three methods of locating the SMRAM in system memory will guarantee cache coherency:

- Place the SRAM in a dedicated section of system memory that the operating system and applications are prevented from accessing. Here, the SRAM can be designated as

1. The VM-exit instruction-length field is not defined following APIC-access VM exits resulting from physical accesses (see Section 25.2.3) even if encountered during delivery of a software interrupt, privileged software exception, or software exception.



cacheable (WB, WT, or WC) for optimum processor performance, without risking cache incoherence when entering or exiting SMM.

- Place the SRAM in a section of memory that overlaps an area used by the operating system (such as the video memory), but designate the SMRAM as uncacheable (UC). This method prevents cache access when in SMM to maintain cache coherency, but the use of uncacheable memory reduces the performance of SMM code.
- Place the SRAM in a section of system memory that overlaps an area used by the operating system and/or application code, but explicitly flush (write back and invalidate) the caches upon entering and exiting SMM mode. This method maintains cache coherency, but incurs the overhead of two complete cache flushes.

...

33.5 SMI HANDLER EXECUTION ENVIRONMENT

After saving the current context of the processor, the processor initializes its core registers to the values shown in Table 33-4. Upon entering SMM, the PE and PG flags in control register CR0 are cleared, which places the processor 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.)
- The normal 64-KByte segment limit for real-address mode is increased to 4 GBytes.
- The default operand and address sizes are set to 16 bits, which restricts the addressable SMRAM address space to the 1-MByte real-address mode limit for native real-address-mode code. However, operand-size and address-size override prefixes can be used to access the address space beyond the 1-MByte.

...