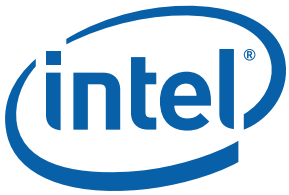


Intel® 64 and IA-32 Architectures Software Developer's Manual

Documentation Changes

May 2011

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: 252046-032

Contents

Revision History

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

Revision	Description	Date
-024	• Removed Documentation Changes 1-21 • Added Documentation Changes 1-16	June 2009
-025	• Removed Documentation Changes 1-16 • Added Documentation Changes 1-18	September 2009
-026	• Removed Documentation Changes 1-18 • Added Documentation Changes 1-15	December 2009
-027	• Removed Documentation Changes 1-15 • Added Documentation Changes 1-24	March 2010
-028	• Removed Documentation Changes 1-24 • Added Documentation Changes 1-29	June 2010
-029	• Removed Documentation Changes 1-29 • Added Documentation Changes 1-29	September 2010
-030	• Removed Documentation Changes 1-29 • Added Documentation Changes 1-29	January 2011
-031	• Removed Documentation Changes 1-29 • Added Documentation Changes 1-29	April 2011
-032	• Removed Documentation Changes 1-29 • Added Documentation Changes 1-14	May 2011

§

Preface

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

Affected Documents

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

Nomenclature

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

Summary Tables of Changes

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

Codes Used in Summary Tables

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

Documentation Changes

No.	DOCUMENTATION CHANGES
1	Updates to Chapter 3, Volume 1
2	Updates to Chapter 7, Volume 1
3	Updates to Chapter 3, Volume 2A
4	Updates to Chapter 4, Volume 2B
5	Updates to Appendix A, Volume 2B
6	Updates to Chapter 2, Volume 3A
7	Updates to Chapter 4, Volume 3A
8	Updates to Chapter 8, Volume 3A
9	Updates to Chapter 10, Volume 3A
10	Updates to Chapter 11, Volume 3A
11	Updates to Chapter 16, Volume 3A
12	Updates to Chapter 28, Volume 3B
13	Updates to Appendix A, Volume 3B
14	Updates to Appendix B, Volume 3B

Documentation Changes

1. Updates to Chapter 3, 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.

...

3.4.1.1 General-Purpose Registers in 64-Bit Mode

In 64-bit mode, there are 16 general purpose registers and the default operand size is 32 bits. However, general-purpose registers are able to work with either 32-bit or 64-bit operands. If a 32-bit operand size is specified: EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D - R15D are available. If a 64-bit operand size is specified: RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8-R15 are available. R8D-R15D/R8-R15 represent eight new general-purpose registers. All of these registers can be accessed at the byte, word, dword, and qword level. REX prefixes are used to generate 64-bit operand sizes or to reference registers R8-R15.

Registers only available in 64-bit mode (R8-R15 and XMM8-XMM15) are preserved across transitions from 64-bit mode into compatibility mode then back into 64-bit mode. However, values of R8-R15 and XMM8-XMM15 are undefined after transitions from 64-bit mode through compatibility mode to legacy or real mode and then back through compatibility mode to 64-bit mode.

Table 3-2 Addressable General Purpose Registers

Register Type	Without REX	With REX
Byte Registers	AL, BL, CL, DL, AH, BH, CH, DH	AL, BL, CL, DL, DIL, SIL, BPL, SPL, R8L - R15L
Word Registers	AX, BX, CX, DX, DI, SI, BP, SP	AX, BX, CX, DX, DI, SI, BP, SP, R8W - R15W
Doubleword Registers	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D - R15D
Quadword Registers	N.A.	RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8 - R15

In 64-bit mode, there are limitations on accessing byte registers. An instruction cannot reference legacy high-bytes (for example: AH, BH, CH, DH) and one of the new byte registers at the same time (for example: the low byte of the RAX register). However, instructions may reference legacy low-bytes (for example: AL, BL, CL or DL) and new byte registers at the same time (for example: the low byte of the R8 register, or RBP). The architecture enforces this limitation by changing high-byte references (AH, BH, CH, DH) to low byte references (BPL, SPL, DIL, SIL: the low 8 bits for RBP, RSP, RDI and RSI) for instructions using a REX prefix.

When in 64-bit mode, operand size determines the number of valid bits in the destination general-purpose register:

- 64-bit operands generate a 64-bit result in the destination general-purpose register.
- 32-bit operands generate a 32-bit result, zero-extended to a 64-bit result in the destination general-purpose register.
- 8-bit and 16-bit operands generate an 8-bit or 16-bit result. The upper 56 bits or 48 bits (respectively) of the destination general-purpose register are not modified by the operation. If the result of an 8-bit or 16-bit operation is intended for 64-bit address calculation, explicitly sign-extend the register to the full 64-bits.

Because the upper 32 bits of 64-bit general-purpose registers are undefined in 32-bit modes, the upper 32 bits of any general-purpose register are not preserved when switching from 64-bit mode to a 32-bit mode (to protected mode or compatibility mode). Software must not depend on these bits to maintain a value after a 64-bit to 32-bit mode switch.

...

2. Updates to Chapter 7, Volume 1

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

...

7.3.18 Random Number Generator Instruction

The RDRAND instruction can provide software with sequences of random numbers generated from white noise.

Truly random numbers can help programmers improve the security of software agents running in a system. The RDRAND instruction provides a facility for programmers to achieve that goal. All Intel processors that support the RDRAND instruction indicate the availability of the RDRAND instruction via reporting CPUID.01H:ECX.RDRAND[bit 30] = 1.

The random numbers that are returned by the RDRAND instruction are supplied by a cryptographically secure Random Number Generator that employs a hardware DRBG (Digital Random Bit Generator, also known as a Pseudo Random Number Generator) seeded by a hardware NRBG (Nondeterministic Random Bit Generator, also known as a TRNG or True Random Number generator).

In order for the hardware design to meet its security goals, the random number generator continuously tests itself and the random data it is generating. Runtime failures in the random number generator circuitry or statistically anomalous data occurring by chance will be detected by the self test hardware and flag the resulting data as being bad. In such extremely rare cases, the RDRAND instruction will return no data instead of bad data.

Under heavy load, with multiple cores executing RDRAND in parallel, it is possible, though unlikely, for the demand of random numbers by software processes/threads to exceed the rate at which the random number generator hardware can supply them. This will lead to the RDRAND instruction returning no data transitorily. The RDRAND instruction indicates the occurrence of this rare situation by clearing the CF flag.

The RDRAND instruction returns with the carry flag set (CF = 1) to indicate data was returned. Software using the RDRAND instruction to get random numbers should retry for a limited number of iterations while RDRAND returns CF=0 and should complete when data is returned, indicated with CF=1. This will deal with transitory underflows. A retry limit should be employed to prevent a hard failure in the RNG (expected to be extremely rare) leading to a busy loop in software.

The intrinsic primitive for RDRAND is defined to address software's need for the common cases (CF = 1) and the rare situations (CF = 0). The intrinsic primitive returns a value that reflects the value of the carry flag returned by the underlying RDRAND instruction. The example below illustrates the recommended usage of an RDRAND instrinsic in a utility function, a loop to fetch a 64 bit random value with a retry count limit of 10. A C implementation might be written as follows:

```
-------------------------------------------------------------------------------------
#define SUCCESS 1
#define RETRY_LIMIT_EXCEEDED 0
#define RETRY_LIMIT 10

int get_random_64( unsigned __int 64 * arand)
{int i ;
    for ( i = 0; i < RETRY_LIMIT; i ++) {
        if(_rdrand64_step(arand) ) return SUCCESS;
    }
    return RETRY_LIMIT_EXCEEDED;
}
-------------------------------------------------------------------------------
```

...

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-M.

...

3.1.1.6 CPUID Support Column in the Instruction Summary Table

The fourth column holds abbreviated CPUID feature flags (e.g. appropriate bit in CPUID.1.ECX, CPUID.1.EDX for SSE/SSE2/SSE3/SSSE3/SSE4.1/SSE4.2/AESNI/ PCLMULQDQ/AVX/RDRAND support) that indicate processor support for the instruction. If the corresponding flag is '0', the instruction will #UD.

...

CPUID—CPU Identification

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
0F A2	CPUID	A	Valid	Valid	Returns processor identification and feature information to the EAX, EBX, ECX, and EDX registers, as determined by input entered in EAX (in some cases, ECX as well).

Instruction Operand Encoding

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

Description

The ID flag (bit 21) in the EFLAGS register indicates support for the CPUID instruction. If a software procedure can set and clear this flag, the processor executing the procedure supports the CPUID instruction. This instruction operates the same in non-64-bit modes and 64-bit mode.

CPUID returns processor identification and feature information in the EAX, EBX, ECX, and EDX registers.[1] The instruction's output is dependent on the contents of the EAX register upon execution (in some cases, ECX as well). For example, the following pseudocode loads EAX with 00H and causes CPUID to return a Maximum Return Value and the Vendor Identification String in the appropriate registers:

```
MOV EAX, 00H
CPUID
```

1. On Intel 64 processors, CPUID clears the high 32 bits of the RAX/RBX/RCX/RDX registers in all modes.

Table 3-17 shows information returned, depending on the initial value loaded into the EAX register. Table 3-18 shows the maximum CPUID input value recognized for each family of IA-32 processors on which CPUID is implemented.

Two types of information are returned: basic and extended function information. If a value entered for CPUID.EAX is higher than the maximum input value for basic or extended function for that processor then the data for the highest basic information leaf is returned. For example, using the Intel Core i7 processor, the following is true:

CPUID.EAX = 05H (* Returns MONITOR/MWAIT leaf. *)
CPUID.EAX = 0AH (* Returns Architectural Performance Monitoring leaf. *)
CPUID.EAX = 0BH (* Returns Extended Topology Enumeration leaf. *)
CPUID.EAX = 0CH (* INVALID: Returns the same information as CPUID.EAX = 0BH. *)
CPUID.EAX = 80000008H (* Returns linear/physical address size data. *)
CPUID.EAX = 8000000AH (* INVALID: Returns same information as CPUID.EAX = 0BH. *)

If a value entered for CPUID.EAX is less than or equal to the maximum input value and the leaf is not supported on that processor then 0 is returned in all the registers. For example, using the Intel Core i7 processor, the following is true:

CPUID.EAX = 07H (*Returns EAX=EBX=ECX=EDX=0. *)

When CPUID returns the highest basic leaf information as a result of an invalid input EAX value, any dependence on input ECX value in the basic leaf is honored.

CPUID can be executed at any privilege level to serialize instruction execution. Serializing instruction execution guarantees that any modifications to flags, registers, and memory for previous instructions are completed before the next instruction is fetched and executed.

See also:

"Serializing Instructions" in Chapter 8, "Multiple-Processor Management," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*

"Caching Translation Information" in Chapter 4, "Paging," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Table 3-17 Information Returned by CPUID Instruction

Initial EAX Value	Information Provided about the Processor	
	Basic CPUID Information	
0H	EAX EBX ECX EDX	Maximum Input Value for Basic CPUID Information (see Table 3-18) "Genu" "ntel" "inel"
01H	EAX EBX ECX EDX	Version Information: Type, Family, Model, and Stepping ID (see Figure 3-5) Bits 07-00: Brand Index Bits 15-08: CLFLUSH line size (Value ∗ 8 = cache line size in bytes) Bits 23-16: Maximum number of addressable IDs for logical processors in this physical package*. Bits 31-24: Initial APIC ID Feature Information (see Figure 3-6 and Table 3-20) Feature Information (see Figure 3-7 and Table 3-21)

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
		NOTES: * The nearest power-of-2 integer that is not smaller than EBX[23:16] is the number of unique initial APIC IDs reserved for addressing different logical processors in a physical package.
02H	EAX EBX ECX EDX	Cache and TLB Information (see Table 3-22) Cache and TLB Information Cache and TLB Information Cache and TLB Information
03H	EAX EBX ECX EDX	Reserved. Reserved. Bits 00-31 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.) Bits 32-63 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.) **NOTES:** Processor serial number (PSN) is not supported in the Pentium 4 processor or later. On all models, use the PSN flag (returned using CPUID) to check for PSN support before accessing the feature. See AP-485, *Intel Processor Identification and the CPUID Instruction* (Order Number 241618) for more information on PSN.
	CPUID leaves > 3 < 80000000 are visible only when IA32_MISC_ENABLE.BOOT_NT4[bit 22] = 0 (default).	
	Deterministic Cache Parameters Leaf	
04H		**NOTES:** Leaf 04H output depends on the initial value in ECX. See also: "INPUT EAX = 4: Returns Deterministic Cache Parameters for each level on page 3-224.
	EAX	Bits 04-00: Cache Type Field 0 = Null - No more caches 1 = Data Cache 2 = Instruction Cache 3 = Unified Cache 4-31 = Reserved Bits 07-05: Cache Level (starts at 1) Bit 08: Self Initializing cache level (does not need SW initialization) Bit 09: Fully Associative cache Bits 13-10: Reserved Bits 25-14: Maximum number of addressable IDs for logical processors sharing this cache*, ** Bits 31-26: Maximum number of addressable IDs for processor cores in the physical package*, ***, ****

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EBX	Bits 11-00: L = System Coherency Line Size* Bits 21-12: P = Physical Line partitions* Bits 31-22: W = Ways of associativity*
	ECX	Bits 31-00: S = Number of Sets*
	EDX	Bit 0: Write-Back Invalidate/Invalidate 0 = WBINVD/INVD from threads sharing this cache acts upon lower level caches for threads sharing this cache. 1 = WBINVD/INVD is not guaranteed to act upon lower level caches of non-originating threads sharing this cache. Bit 1: Cache Inclusiveness 0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels. Bit 2: Complex Cache Indexing 0 = Direct mapped cache. 1 = A complex function is used to index the cache, potentially using all address bits. Bits 31-03: Reserved = 0
		NOTES: * Add one to the return value to get the result. ** The nearest power-of-2 integer that is not smaller than (1 + EAX[25:14]) is the number of unique initial APIC IDs reserved for addressing different logical processors sharing this cache *** The nearest power-of-2 integer that is not smaller than (1 + EAX[31:26]) is the number of unique Core_IDs reserved for addressing different processor cores in a physical package. Core ID is a subset of bits of the initial APIC ID. ****The returned value is constant for valid initial values in ECX. Valid ECX values start from 0.
	MONITOR/MWAIT Leaf	
05H	EAX	Bits 15-00: Smallest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	EBX	Bits 15-00: Largest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	ECX	Bit 00: Enumeration of Monitor-Mwait extensions (beyond EAX and EBX registers) supported Bit 01: Supports treating interrupts as break-event for MWAIT, even when interrupts disabled Bits 31 - 02: Reserved

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 03 - 00: Number of C0* sub C-states supported using MWAIT Bits 07 - 04: Number of C1* sub C-states supported using MWAIT Bits 11 - 08: Number of C2* sub C-states supported using MWAIT Bits 15 - 12: Number of C3* sub C-states supported using MWAIT Bits 19 - 16: Number of C4* sub C-states supported using MWAIT Bits 31 - 20: Reserved = 0 **NOTE:** * The definition of C0 through C4 states for MWAIT extension are processor-specific C-states, not ACPI C-states.
	Thermal and Power Management Leaf	
06H	EAX	Bit 00: Digital temperature sensor is supported if set Bit 01: Intel Turbo Boost Technology Available (see description of IA32_MISC_ENABLE[38]). Bit 02: ARAT. APIC-Timer-always-running feature is supported if set. Bit 03: Reserved Bit 04: PLN. Power limit notification controls are supported if set. Bit 05: ECMD. Clock modulation duty cycle extension is supported if set. Bit 06: PTM. Package thermal management is supported if set. Bits 31 - 07: Reserved
	EBX	Bits 03 - 00: Number of Interrupt Thresholds in Digital Thermal Sensor Bits 31 - 04: Reserved
	ECX	Bit 00: Hardware Coordination Feedback Capability (Presence of IA32_MPERF and IA32_APERF). The capability to provide a measure of delivered processor performance (since last reset of the counters), as a percentage of expected processor performance at frequency specified in CPUID Brand String Bits 02 - 01: Reserved = 0 Bit 03: The processor supports performance-energy bias preference if CPUID.06H:ECX.SETBH[bit 3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1B0H) Bits 31 - 04: Reserved = 0
	EDX	Reserved = 0
	Structured Extended Feature Flags Enumeration Leaf (Output depends on ECX input value)	
07H		Sub leaf 0 (Input ECX = 0).
	EAX	Bits 31-00: Reports the maximum number of supported leaf 7 sub-leaves.
	EBX	Bit 00: Supports RDFSBASE/RDGSBASE/WRFSBASE/WRGSBASE if 1. Bit 06:01: Reserved Bit 07: Supports Supervisor Mode Execution Protection (SMEP) if 1 Bit 08: Reserved Bit 09: Supports Enhanced REP MOVSB/STOSB if 1. Bit 31:10: Reserved
	ECX	Reserved
	EDX	Reserved.

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	Direct Cache Access Information Leaf	
09H	EAX	Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H)
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
	Architectural Performance Monitoring Leaf	
0AH	EAX	Bits 07 - 00: Version ID of architectural performance monitoring Bits 15- 08: Number of general-purpose performance monitoring counter per logical processor Bits 23 - 16: Bit width of general-purpose, performance monitoring counter Bits 31 - 24: Length of EBX bit vector to enumerate architectural performance monitoring events
	EBX	Bit 00: Core cycle event not available if 1 Bit 01: Instruction retired event not available if 1 Bit 02: Reference cycles event not available if 1 Bit 03: Last-level cache reference event not available if 1 Bit 04: Last-level cache misses event not available if 1 Bit 05: Branch instruction retired event not available if 1 Bit 06: Branch mispredict retired event not available if 1 Bits 31- 07: Reserved = 0
	ECX	Reserved = 0
	EDX	Bits 04 - 00: Number of fixed-function performance counters (if Version ID > 1) Bits 12- 05: Bit width of fixed-function performance counters (if Version ID > 1) Reserved = 0
	Extended Topology Enumeration Leaf	
0BH		**NOTES:** Most of Leaf 0BH output depends on the initial value in ECX. EDX output do not vary with initial value in ECX. ECX[7:0] output always reflect initial value in ECX. All other output value for an invalid initial value in ECX are 0. Leaf 0BH exists if EBX[15:0] is not zero.
	EAX	Bits 04-00: Number of bits to shift right on x2APIC ID to get a unique topology ID of the next level type*. All logical processors with the same next level ID share current level. Bits 31-05: Reserved.
	EBX	Bits 15 - 00: Number of logical processors at this level type. The number reflects configuration as shipped by Intel**. Bits 31- 16: Reserved.

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bits 07 - 00: Level number. Same value in ECX input Bits 15 - 08: Level type***. Bits 31 - 16:: Reserved.
	EDX	Bits 31- 00: x2APIC ID the current logical processor. **NOTES:** * Software should use this field (EAX[4:0]) to enumerate processor topology of the system. ** Software must not use EBX[15:0] to enumerate processor topology of the system. This value in this field (EBX[15:0]) is only intended for display/diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations. *** The value of the "level type" field is not related to level numbers in any way, higher "level type" values do not mean higher levels. Level type field has the following encoding: 0 : invalid 1 : SMT 2 : Core 3-255 : Reserved
Processor Extended State Enumeration Main Leaf (EAX = 0DH, ECX = 0)		
0DH		**NOTES:** Leaf 0DH main leaf (ECX = 0).
	EAX	Bits 31-00: Reports the valid bit fields of the lower 32 bits of XCR0. If a bit is 0, the corresponding bit field in XCR0 is reserved. Bit 00: legacy x87 Bit 01: 128-bit SSE Bit 02: 256-bit AVX Bits 31- 03: Reserved
	EBX	Bits 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than ECX if some features at the end of the XSAVE save area are not enabled.
	ECX	Bit 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processor, i.e all the valid bit fields in XCR0.
	EDX	Bit 31-00: Reports the valid bit fields of the upper 32 bits of XCR0. If a bit is 0, the corresponding bit field in XCR0 is reserved.
Processor Extended State Enumeration Sub-leaf (EAX = 0DH, ECX = 1)		

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EAX	Bits 31-01: Reserved Bit 00: XSAVEOPT is available;
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
Processor Extended State Enumeration Sub-leaves (EAX = 0DH, ECX = n, n > 1)		
0DH		**NOTES:** Leaf 0DH output depends on the initial value in ECX. If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0. Each valid sub-leaf index maps to a valid bit in the XCR0 register starting at bit position 2
	EAX	Bits 31-0: The size in bytes (from the offset specified in EBX) of the save area for an extended state feature associated with a valid sub-leaf index, *n*. This field reports 0 if the sub-leaf index, *n*, is invalid*.
	EBX	Bits 31-0: The offset in bytes of this extended state component's save area from the beginning of the XSAVE/XRSTOR area. This field reports 0 if the sub-leaf index, *n*, is invalid*.
	ECX	This field reports 0 if the sub-leaf index, *n*, is invalid*; otherwise it is reserved.
	EDX	This field reports 0 if the sub-leaf index, *n*, is invalid*; otherwise it is reserved.
Unimplemented CPUID Leaf Functions		
40000000H - 4FFFFFFFH		Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is in the range 40000000H to 4FFFFFFFH.
Extended Function CPUID Information		
80000000H	EAX	Maximum Input Value for Extended Function CPUID Information (see Table 3-18).
	EBX ECX EDX	Reserved Reserved Reserved
80000001H	EAX	Extended Processor Signature and Feature Bits.
	EBX	Reserved
	ECX	Bit 00: LAHF/SAHF available in 64-bit mode Bits 31-01 Reserved

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 10-00: Reserved Bit 11: SYSCALL/SYSRET available (when in 64-bit mode) Bits 19-12: Reserved = 0 Bit 20: Execute Disable Bit available Bits 25-21: Reserved = 0 Bit 26: 1-GByte pages are available if 1 Bit 27: RDTSCP and IA32_TSC_AUX are available if 1 Bits 28: Reserved = 0 Bit 29: Intel® 64 Architecture available if 1 Bits 31-30: Reserved = 0
80000002H	EAX EBX ECX EDX	Processor Brand String Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued
80000003H	EAX EBX ECX EDX	Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued
80000004H	EAX EBX ECX EDX	Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued
80000005H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 Reserved = 0
80000006H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Bits 07-00: Cache Line size in bytes Bits 11-08: Reserved Bits 15-12: L2 Associativity field * Bits 31-16: Cache size in 1K units Reserved = 0
		NOTES: * L2 associativity field encodings: 00H - Disabled 01H - Direct mapped 02H - 2-way 04H - 4-way 06H - 8-way 08H - 16-way 0FH - Fully associative

Table 3-17 Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
80000007H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 Bits 07-00: Reserved = 0 Bit 08: Invariant TSC available if 1 Bits 31-09: Reserved = 0
80000008H	EAX	Linear/Physical Address size Bits 07-00: #Physical Address Bits* Bits 15-8: #Linear Address Bits Bits 31-16: Reserved = 0
	EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0
		NOTES: * If CPUID.80000008H:EAX[7:0] is supported, the maximum physical address number supported should come from this field.

…

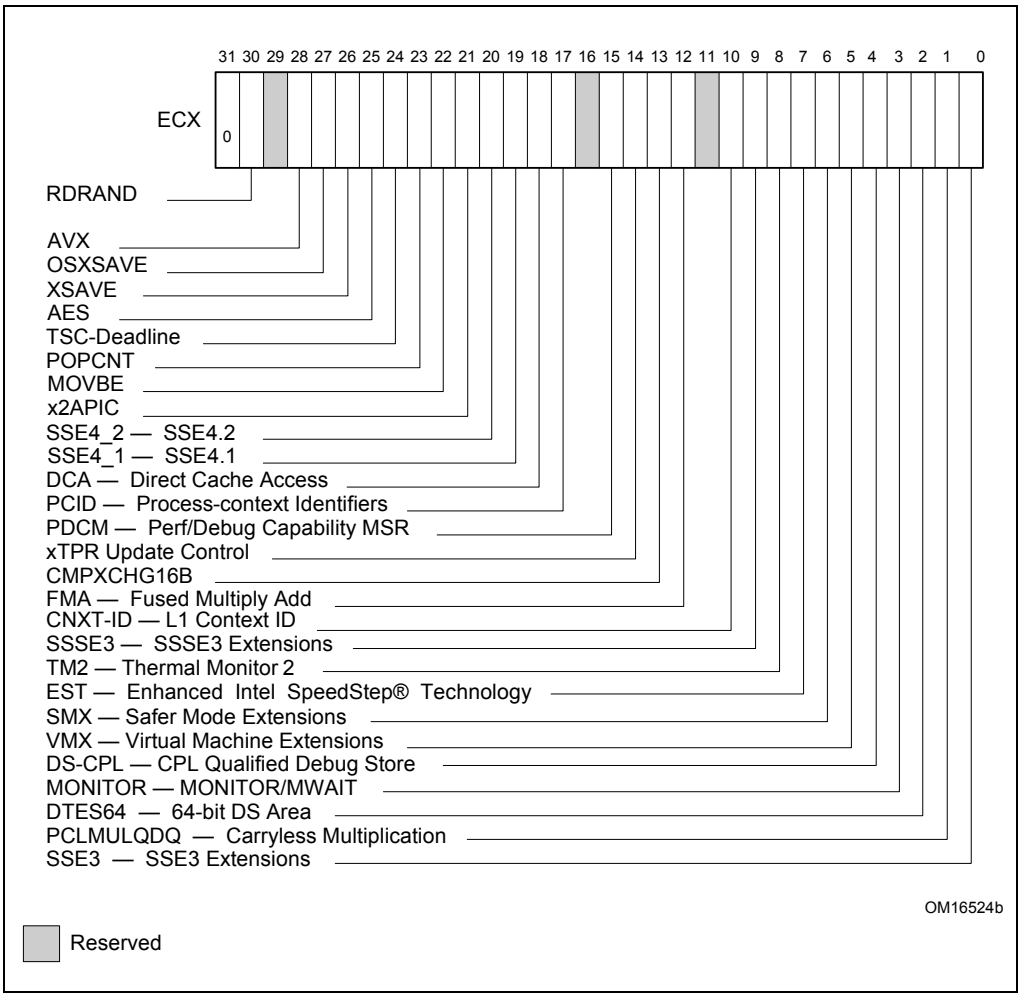


Figure 3-6 Feature Information Returned in the ECX Register

Table 3-20 Feature Information Returned in the ECX Register

Bit #	Mnemonic	Description
0	SSE3	**Streaming SIMD Extensions 3 (SSE3).** A value of 1 indicates the processor supports this technology.
1	PCLMULQDQ	**PCLMULQDQ.** A value of 1 indicates the processor supports the PCLMULQDQ instruction
2	DTES64	**64-bit DS Area.** A value of 1 indicates the processor supports DS area using 64-bit layout
3	MONITOR	**MONITOR/MWAIT.** A value of 1 indicates the processor supports this feature.
4	DS-CPL	**CPL Qualified Debug Store.** A value of 1 indicates the processor supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.

Table 3-20 Feature Information Returned in the ECX Register (Contd.)

Bit #	Mnemonic	Description
5	VMX	**Virtual Machine Extensions.** A value of 1 indicates that the processor supports this technology
6	SMX	**Safer Mode Extensions.** A value of 1 indicates that the processor supports this technology. See Chapter 6, "Safer Mode Extensions Reference".
7	EIST	**Enhanced Intel SpeedStep® technology.** A value of 1 indicates that the processor supports this technology.
8	TM2	**Thermal Monitor 2.** A value of 1 indicates whether the processor supports this technology.
9	SSSE3	A value of 1 indicates the presence of the Supplemental Streaming SIMD Extensions 3 (SSSE3). A value of 0 indicates the instruction extensions are not present in the processor
10	CNXT-ID	**L1 Context ID.** A value of 1 indicates the L1 data cache mode can be set to either adaptive mode or shared mode. A value of 0 indicates this feature is not supported. See definition of the IA32_MISC_ENABLE MSR Bit 24 (L1 Data Cache Context Mode) for details.
11	Reserved	Reserved
12	FMA	A value of 1 indicates the processor supports FMA extensions using YMM state.
13	CMPXCHG16B	**CMPXCHG16B Available.** A value of 1 indicates that the feature is available. See the "CMPXCHG8B/CMPXCHG16B—Compare and Exchange Bytes" section in this chapter for a description.
14	xTPR Update Control	**xTPR Update Control.** A value of 1 indicates that the processor supports changing IA32_MISC_ENABLE[bit 23].
15	PDCM	**Perfmon and Debug Capability:** A value of 1 indicates the processor supports the performance and debug feature indication MSR IA32_PERF_CAPABILITIES.
16	Reserved	Reserved
17	PCID	**Process-context identifiers.** A value of 1 indicates that the processor supports PCIDs and that software may set CR4.PCIDE to 1.
18	DCA	A value of 1 indicates the processor supports the ability to prefetch data from a memory mapped device.
19	SSE4.1	A value of 1 indicates that the processor supports SSE4.1.
20	SSE4.2	A value of 1 indicates that the processor supports SSE4.2.
21	x2APIC	A value of 1 indicates that the processor supports x2APIC feature.
22	MOVBE	A value of 1 indicates that the processor supports MOVBE instruction.
23	POPCNT	A value of 1 indicates that the processor supports the POPCNT instruction.
24	TSC-Deadline	A value of 1 indicates that the processor's local APIC timer supports one-shot operation using a TSC deadline value.
25	AESNI	A value of 1 indicates that the processor supports the AESNI instruction extensions.

Table 3-20 Feature Information Returned in the ECX Register (Contd.)

Bit #	Mnemonic	Description
26	XSAVE	A value of 1 indicates that the processor supports the XSAVE/XRSTOR processor extended states feature, the XSETBV/XGETBV instructions, and XCR0.
27	OSXSAVE	A value of 1 indicates that the OS has enabled XSETBV/XGETBV instructions to access XCR0, and support for processor extended state management using XSAVE/XRSTOR.
28	AVX	A value of 1 indicates the processor supports the AVX instruction extensions.
29	Reserved	Reserved
30	RDRAND	A value of 1 indicates that processor supports RDRAND instruction.
31	Not Used	Always returns 0

...

INPUT EAX = 07H: Returns Structured Extended Feature Enumeration Information

When CPUID executes with EAX set to 7 and ECX = 0, the processor returns information about the maximum number of sub-leaves that contain extended feature flags. See Table 3-17.

When CPUID executes with EAX set to 7 and ECX = n (n > 1and less than the number of non-zero bits in CPUID.(EAX=07H, ECX= 0H).EAX, the processor returns information about extended feature flags. See Table 3-17. In subleaf 0, only EAX has the number of subleafs. In subleaf 0, EBX, ECX & EDX all contain extended feature flags.

...

IA-32 Architecture Compatibility

CPUID is not supported in early models of the Intel486 processor or in any IA-32 processor earlier than the Intel486 processor.

Operation

```
IA32_BIOS_SIGN_ID MSR ← Update with installed microcode revision number;

CASE (EAX) OF
    EAX = 0:
        EAX ← Highest basic function input value understood by CPUID;
        EBX ← Vendor identification string;
        EDX ← Vendor identification string;
        ECX ← Vendor identification string;
    BREAK;
    EAX = 1H:
        EAX[3:0] ← Stepping ID;
        EAX[7:4] ← Model;
        EAX[11:8] ← Family;
        EAX[13:12] ← Processor type;
        EAX[15:14] ← Reserved;
```

```
        EAX[19:16] ← Extended Model;
        EAX[27:20] ← Extended Family;
        EAX[31:28] ← Reserved;
        EBX[7:0] ← Brand Index; (* Reserved if the value is zero. *)
        EBX[15:8] ← CLFLUSH Line Size;
        EBX[16:23] ← Reserved; (* Number of threads enabled = 2 if MT enable fuse set. *)
        EBX[24:31] ← Initial APIC ID;
        ECX ← Feature flags; (* See Figure 3-6. *)
        EDX ← Feature flags; (* See Figure 3-7. *)
BREAK;
EAX = 2H:
        EAX ← Cache and TLB information;
        EBX ← Cache and TLB information;
        ECX ← Cache and TLB information;
        EDX ← Cache and TLB information;
BREAK;
EAX = 3H:
        EAX ← Reserved;
        EBX ← Reserved;
        ECX ← ProcessorSerialNumber[31:0];
        (* Pentium III processors only, otherwise reserved. *)
        EDX ← ProcessorSerialNumber[63:32];
        (* Pentium III processors only, otherwise reserved. *
BREAK
EAX = 4H:
        EAX ← Deterministic Cache Parameters Leaf; (* See Table 3-17. *)
        EBX ← Deterministic Cache Parameters Leaf;
        ECX ← Deterministic Cache Parameters Leaf;
        EDX ← Deterministic Cache Parameters Leaf;
BREAK;
EAX = 5H:
        EAX ← MONITOR/MWAIT Leaf; (* See Table 3-17. *)
        EBX ← MONITOR/MWAIT Leaf;
        ECX ← MONITOR/MWAIT Leaf;
        EDX ← MONITOR/MWAIT Leaf;
BREAK;
EAX = 6H:
        EAX ← Thermal and Power Management Leaf; (* See Table 3-17. *)
        EBX ← Thermal and Power Management Leaf;
        ECX ← Thermal and Power Management Leaf;
        EDX ← Thermal and Power Management Leaf;
BREAK;
EAX = 7H:
        EAX ← Structured Extended Feature Flags Enumeration Leaf; (* See Table 3-17. *)
        EBX ← Structured Extended Feature Flags Enumeration Leaf;
        ECX ← Structured Extended Feature Flags Enumeration Leaf;
        EDX ← Structured Extended Feature Flags Enumeration Leaf;
BREAK;
EAX = 8H:
        EAX ← Reserved = 0;
```

```
            EBX ← Reserved = 0;
            ECX ← Reserved = 0;
            EDX ← Reserved = 0;
        BREAK;
        EAX = 9H:
            EAX ← Direct Cache Access Information Leaf; (* See Table 3-17. *)
            EBX ← Direct Cache Access Information Leaf;
            ECX ← Direct Cache Access Information Leaf;
            EDX ← Direct Cache Access Information Leaf;
        BREAK;
        EAX = AH:
            EAX ← Architectural Performance Monitoring Leaf; (* See Table 3-17. *)
            EBX ← Architectural Performance Monitoring Leaf;
            ECX ← Architectural Performance Monitoring Leaf;
            EDX ← Architectural Performance Monitoring Leaf;
            BREAK
        EAX = BH:
            EAX ← Extended Topology Enumeration Leaf; (* See Table 3-17. *)
            EBX ← Extended Topology Enumeration Leaf;
            ECX ← Extended Topology Enumeration Leaf;
            EDX ← Extended Topology Enumeration Leaf;
        BREAK;
        EAX = CH:
            EAX ← Reserved = 0;
            EBX ← Reserved = 0;
            ECX ← Reserved = 0;
            EDX ← Reserved = 0;
        BREAK;
        EAX = DH:
            EAX ← Processor Extended State Enumeration Leaf; (* See Table 3-17. *)
            EBX ← Processor Extended State Enumeration Leaf;
            ECX ← Processor Extended State Enumeration Leaf;
            EDX ← Processor Extended State Enumeration Leaf;
        BREAK;
    BREAK;
    EAX = 80000000H:
        EAX ← Highest extended function input value understood by CPUID;
        EBX ← Reserved;
        ECX ← Reserved;
        EDX ← Reserved;
    BREAK;
    EAX = 80000001H:
        EAX ← Reserved;
        EBX ← Reserved;
        ECX ← Extended Feature Bits (* See Table 3-17.*);
        EDX ← Extended Feature Bits (* See Table 3-17. *);
    BREAK;
    EAX = 80000002H:
        EAX ← Processor Brand String;
        EBX ← Processor Brand String, continued;
```

```
            ECX ← Processor Brand String, continued;
            EDX ← Processor Brand String, continued;
        BREAK;
        EAX = 80000003H:
            EAX ← Processor Brand String, continued;
            EBX ← Processor Brand String, continued;
            ECX ← Processor Brand String, continued;
            EDX ← Processor Brand String, continued;
        BREAK;
        EAX = 80000004H:
            EAX ← Processor Brand String, continued;
            EBX ← Processor Brand String, continued;
            ECX ← Processor Brand String, continued;
            EDX ← Processor Brand String, continued;
        BREAK;
        EAX = 80000005H:
            EAX ← Reserved = 0;
            EBX ← Reserved = 0;
            ECX ← Reserved = 0;
            EDX ← Reserved = 0;
        BREAK;
        EAX = 80000006H:
            EAX ← Reserved = 0;
            EBX ← Reserved = 0;
            ECX ← Cache information;
            EDX ← Reserved = 0;
        BREAK;
        EAX = 80000007H:
            EAX ← Reserved = 0;
            EBX ← Reserved = 0;
            ECX ← Reserved = 0;
            EDX ← Reserved = Misc Feature Flags;
        BREAK;
        EAX = 80000008H:
            EAX ← Reserved = Physical Address Size Information;
            EBX ← Reserved = Virtual Address Size Information;
            ECX ← Reserved = 0;
            EDX ← Reserved = 0;
        BREAK;
        EAX >= 40000000H and EAX <= 4FFFFFFFH:
        DEFAULT: (* EAX = Value outside of recognized range for CPUID. *)
            (* If the highest basic information leaf data depend on ECX input value, ECX is honored.*)
            EAX ← Reserved; (* Information returned for highest basic information leaf. *)
            EBX ← Reserved; (* Information returned for highest basic information leaf. *)
            ECX ← Reserved; (* Information returned for highest basic information leaf. *)
            EDX ← Reserved; (* Information returned for highest basic information leaf. *)
        BREAK;
ESAC;
```

Flags Affected

None.

Exceptions (All Operating Modes)

#UD	If the LOCK prefix is used. In earlier IA-32 processors that do not support the CPUID instruction, execution of the instruction results in an invalid opcode (#UD) exception being generated.

...

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, N-Z.

...

RDRAND—Read Random Number

Opcode*/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F C7 /6 RDRAND r16	A	V/V	RDRAND	Read a 16-bit random number and store in the destination register.
0F C7 /6 RDRAND r32	A	V/V	RDRAND	Read a 32-bit random number and store in the destination register.
REX.W + 0F C7 /6 RDRAND r64	A	V/I	RDRAND	Read a 64-bit random number and store in the destination register.

Instruction Operand Encoding

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

Description

Loads a hardware generated random value and store it in the destination register. The size of the random value is determined by the destination register size and operating mode. The Carry Flag indicates whether a random value is available at the time the instruction is executed. CF=1 indicates that the data in the destination is valid. Otherwise CF=0 and the data in the destination operand will be returned as zeros for the specified width. All other flags are forced to 0 in either situation. Software must check the state of CF=1 for determining if a valid random value has been returned, otherwise it is expected to loop and retry execution of RDRAND (see *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, Section 7.3.18, "Random Number Generator Instruction").

This instruction is available at all privilege levels. For virtualization supporting lockstep operation, a virtualization control exists that allows the virtual machine monitor to trap on the instruction. "RDRAND exiting" will be controlled by bit 11 of the secondary processor-based VM-execution control. A VMEXIT due to RDRAND will have exit reason 57 (decimal).

In 64-bit mode, the instruction's default operation size is 32 bits. Using a REX prefix in the form of REX.B permits access to additional registers (R8-R15). Using a REX prefix in the form of REX.W promotes operation to 64 bit operands. See the summary chart at the beginning of this section for encoding data and limits.

```
    THEN
        CASE of
            osize is 64: DEST[63:0] ← HW_RND_GEN.data;
            osize is 32: DEST[31:0] ← HW_RND_GEN.data;
            osize is 16: DEST[15:0] ← HW_RND_GEN.data;
        ESAC
        CF ← 1;
    ELSE
        CASE of
            osize is 64: DEST[63:0] ← 0;
            osize is 32: DEST[31:0] ← 0;
            osize is 16: DEST[15:0] ← 0;
        ESAC
        CF ← 0;
FI
OF, SF, ZF, AF, PF ← 0;
```

Flags Affected

All flags are affected.

Intel C/C++ Compiler Intrinsic Equivalent

RDRAND int _rdrand16_step(unsigned short *);

RDRAND int _rdrand32_step(unsigned int *);

RDRAND int _rdrand64_step(unsigned __int64 *);

Protected Mode Exceptions

#UD	If the LOCK prefix is used.
	If the F2H or F3H prefix is used.
	If CPUID.01H:ECX.RDRAND[bit 30] = 0.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

...

XRSTOR—Restore Processor Extended States

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

Instruction Operand Encoding

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

Description

Performs a full or partial restore of the enabled processor states using the state information stored in the memory address specified by the source operand. The implicit EDX:EAX register pair specifies a 64-bit restore mask.

The format of the XSAVE/XRSTOR area is shown in Table 4-18. The memory layout of the XSAVE/XRSTOR area may have holes between save areas written by the processor as a result of the processor not supporting certain processor extended states or system software not supporting certain processor extended states. There is no relationship between the order of XCR0 bits and the order of the state layout. States corresponding to higher and lower XCR0 bits may be intermingled in the layout.

Table 4-18 General Layout of XSAVE/XRSTOR Save Area

Save Areas	Offset (Byte)	Size (Bytes)
FPU/SSE SaveArea[1]	0	512
Header	512	64
Reserved (Ext_Save_Area_2)	CPUID.(EAX=0DH, ECX=2):EBX	CPUID.(EAX=0DH, ECX=2):EAX
Reserved(Ext_Save_Area_4)[2]	CPUID.(EAX=0DH, ECX=4):EBX	CPUID.(EAX=0DH, ECX=4):EAX
Reserved(Ext_Save_Area_3)	CPUID.(EAX=0DH, ECX=3):EBX	CPUID.(EAX=0DH, ECX=3):EAX
Reserved(...)	...	...

NOTES:

1. Bytes 464:511 are available for software use. XRSTOR ignores the value contained in bytes 464:511 of an XSAVE SAVE image.
2. State corresponding to higher and lower XCR0 bits may be intermingled in layout.

XRSTOR operates on each subset of the processor state or a processor extended state in one of three ways (depending on the corresponding bit in XCR0 (XFEATURE_ENABLED_MASK register), the restore mask EDX:EAX, and the save mask XSAVE.HEADER.XSTATE_BV in memory):

- Updates the processor state component using the state information stored in the respective save area (see Table 4-18) of the source operand, if the corresponding bit in XCR0, EDX:EAX, and XSAVE.HEADER.XSTATE_BV are all 1.
- Writes certain registers in the processor state component using processor-supplied values (see Table 4-20) without using state information stored in respective save area of the memory region, if the corresponding bit in XCR0 and EDX:EAX are both 1, but the corresponding bit in XSAVE.HEADER.XSTATE_BV is 0.
- The processor state component is unchanged, if the corresponding bit in XCR0 or EDX:EAX is 0.

The format of the header section (XSAVE.HEADER) of the XSAVE/XRSTOR area is shown in Table 4-19.

Table 4-19 XSAVE.HEADER Layout

15 8	7 0	Byte Offset from Header	Byte Offset from XSAVE/XRSTOR Area
Rsrvd (Must be 0)	XSTATE_BV	0	512
Reserved	Rsrvd (Must be 0)	16	528
Reserved	Reserved	32	544
Reserved	Reserved	48	560

If a processor state component is not enabled in XCR0 but the corresponding save mask bit in XSAVE.HEADER.XSTATE_BV is 1, an attempt to execute XRSTOR will cause a #GP(0) exception. Software may specify all 1's in the implicit restore mask EDX:EAX, so that all the enabled processors states in XCR0 are restored from state information stored in memory or from processor supplied values. When using all 1's as the restore mask, software is required to determine the total size of the XSAVE/XRSTOR save area (specified as source operand) to fit all enabled processor states by using the value enumerated in CPUID.(EAX=0D, ECX=0):EBX. While it's legal to set any bit in the EDX:EAX mask to 1, it is strongly recommended to set only the bits that are required to save/restore specific states.

An attempt to restore processor states with writing 1s to reserved bits in certain registers (see Table 4-21) will cause a #GP(0) exception.

Because bit 63 of XCR0 is reserved for future bit vector expansion, it will not be used for any future processor state feature, and XRSTOR will ignore bit 63 of EDX:EAX (EDX[31]).

...

5. Updates to Appendix A, Volume 2B

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

...

A.2.1 Codes for Addressing Method

The following abbreviations are used to document addressing methods:

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

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

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

E A ModR/M byte follows the opcode and specifies the operand. The operand is either a general-purpose register or a memory address. If it is a memory address, the address is computed from a segment register and any of the

following values: a base register, an index register, a scaling factor, a displacement.

F EFLAGS/RFLAGS Register.

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

H The VEX.vvvv field of the VEX prefix selects a 128-bit XMM register or a 256-bit YMM register, determined by operand type. For legacy SSE encodings this operand does not exist, changing the instruction to destructive form.

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

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

L The upper 4 bits of the 8-bit immediate selects a 128-bit XMM register or a 256-bit YMM register, determined by operand type. (the MSB is ignored in 32-bit mode)

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

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

O The instruction has no ModR/M byte. The offset of the operand is coded as a word or double word (depending on address size attribute) in the instruction. No base register, index register, or scaling factor can be applied (for example, MOV (A0–A3)).

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

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

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

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

U The R/M field of the ModR/M byte selects a 128-bit XMM register or a 256-bit YMM register, determined by operand type.

V The reg field of the ModR/M byte selects a 128-bit XMM register or a 256-bit YMM register, determined by operand type.

W A ModR/M byte follows the opcode and specifies the operand. The operand is either a 128-bit XMM register, a 256-bit YMM register (determined by operand type), or a memory address. If it is a memory address, the address is computed from a segment register and any of the following values: a base register, an index register, a scaling factor, and a displacement.

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

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

A.2.2 Codes for Operand Type

The following abbreviations are used to document operand types:

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

b Byte, regardless of operand-size attribute.

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

d Doubleword, regardless of operand-size attribute.

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

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

pd 128-bit or 256-bit packed double-precision floating-point data.

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

ps 128-bit or 256-bit packed single-precision floating-point data.

q Quadword, regardless of operand-size attribute.

qq Quad-Quadword (256-bits), regardless of operand-size attribute.

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

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

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

si Doubleword integer register (for example: eax).

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

w Word, regardless of operand-size attribute.

x dq or qq based on the operand-size attribute.

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

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

...

A.2.4.4 VEX Prefix Instructions

Instructions that include a VEX prefix are organized relative to the 2-byte and 3-byte opcode maps, based on the VEX.mmmmm field encoding of implied 0F, 0F38H, 0F3AH, respectively. Each entry in the opcode map of a VEX-encoded instruction is based on the value of the opcode byte, similar to non-VEX-encoded instructions.

A VEX prefix includes several bit fields that encode implied 66H, F2H, F3H prefix functionality (VEX.pp) and operand size/opcode information (VEX.L). See chapter 4 for details.

Opcode tables A2-A6 include both instructions with a VEX prefix and instructions without a VEX prefix. Many entries are only made once, but represent both the VEX and non-VEX

forms of the instruction. If the VEX prefix is present all the operands are valid and the mnemonic is usually prefixed with a “v”. If the VEX prefix is not present the VEX.vvvv operand is not available and the prefix “v” is dropped from the mnemonic.

A few instructions exist only in VEX form and these are marked with a superscript “v”.

Operand size of VEX prefix instructions can be determined by the operand type code. 128-bit vectors are indicated by 'dq', 256-bit vectors are indicated by 'qq', and instructions with operands supporting either 128 or 256-bit, determined by VEX.L, are indicated by 'x'. For example, the entry "VMOVUPD Vx,Wx" indicates both VEX.L=0 and VEX.L=1 are supported.

...

A.2.5 Superscripts Utilized in Opcode Tables

Table A-1 contains notes on particular encodings. These notes are indicated in the following opcode maps by superscripts. Gray cells indicate instruction groupings.

Table A-1 Superscripts Utilized in Opcode Tables

Superscript Symbol	Meaning of Symbol
1A	Bits 5, 4, and 3 of ModR/M byte used as an opcode extension (refer to Section A.4, “Opcode Extensions For One-Byte And Two-byte Opcodes”).
1B	Use the 0F0B opcode (UD2 instruction) or the 0FB9H opcode when deliberately trying to generate an invalid opcode exception (#UD).
1C	Some instructions use the same two-byte opcode. If the instruction has variations, or the opcode represents different instructions, the ModR/M byte will be used to differentiate the instruction. For the value of the ModR/M byte needed to decode the instruction, see Table A-6.
i64	The instruction is invalid or not encodable in 64-bit mode. 40 through 4F (single-byte INC and DEC) are REX prefix combinations when in 64-bit mode (use FE/FF Grp 4 and 5 for INC and DEC).
o64	Instruction is only available when in 64-bit mode.
d64	When in 64-bit mode, instruction defaults to 64-bit operand size and cannot encode 32-bit operand size.
f64	The operand size is forced to a 64-bit operand size when in 64-bit mode (prefixes that change operand size are ignored for this instruction in 64-bit mode).
v	VEX form only exists. There is no legacy SSE form of the instruction.
v1	VEX128 & SSE forms only exist (no VEX256), when can’t be inferred from the data size.

...

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

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

NOTES:

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

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

	pfx	0	1	2	3	4	5	6	7
0		Grp 6[1A]	Grp 7[1A]	LAR Gv, Ew	LSL Gv, Ew		SYSCALL[o64]	CLTS	SYSRET[o64]
1		vmovups Vps, Wps	vmovups Wps, Vps	vmovlps Vq, Hq, Mq vmovhlps Vq, Hq, Uq	vmovlps Mq, Vq	vunpcklps Vps, Wq Vx, Hx, Wx	vunpckhps Vps, Wq Vx, Hx, Wx	vmovhps[v1] Vdq, Hq, Mq vmovlhps Vdq, Hq, Uq	vmovhps[v1] Mq, Vq
	66	vmovupd Vpd, Wpd	vmovupd Wpd,Vpd	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 Vss, Wss Vss, Hss, Uss	vmovss Wss, Vss Uss, Hss, Vss	vmovsldup Vx, Wx				vmovshdup Vx, Wx	
	F2	vmovsd Vsd, Wsd Usd, Hsd, Vsd	vmovsd Vsd, Wsd Usd, Hsd, Vsd	vmovddup Vx, Wx					
2	2	MOV Rd, Cd	MOV Rd, Dd	MOV Cd, Rd	MOV Dd, Rd				
3	3	WRMSR	RDTSC	RDMSR	RDPMC	SYSENTER	SYSEXIT		GETSEC
4	4	CMOVcc, (Gv, Ev) - Conditional Move O	NO	B/C/NAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
5		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 Vdq, Hdq, Wdq	vpunpcklwd Vdq, Hdq, Wdq	vpunpckldq Vdq, Hdq, Wdq	vpacksswb Vdq, Hdq,Wdq	vpcmpgtb Vdq, Hdq, Wdq	vpcmpgtw Vdq, Hdq, Wdq	vpcmpgtd Vdq, Hdq, Wdq	vpackuswb Vdq, Hdq, Wdq
	F3								
7		pshufw Pq, Qq, Ib	(Grp 12[1A])	(Grp 13[1A])	(Grp 14[1A])	pcmpeqb Pq, Qq	pcmpeqw Pq, Qq	pcmpeqd Pq, Qq	emms vzeroupper[v] vzeroall[v]
	66	vpshufd Vdq,Wdq,Ib				vpcmpeqb Vdq, Hdq, Wdq	vpcmpeqw Vdq, Hdq, Wdq	vpcmpeqd Vdq, Hdq, Wdq	
	F3	vpshufhw Vdq,Wdq,Ib							
	F2	vpshuflw Vdq,Wdq,Ib							

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

	pfx	8	9	A	B	C	D	E	F
0		INVD	WBINVD		2-byte Illegal Opcodes UD2[1B]		NOP Ev		
1		Prefetch[1C] (Grp 16[1A])							NOP Ev
2		vmovaps Vps, Wps	vmovaps Wps, Vps	cvtpi2ps Vps, Qpi	vmovntps Mps, Vps	cvttps2pi Ppi, Wps	cvtps2pi Ppi, Wps	vucomiss Vss, Wss	vcomiss Vss, Wss
	66	vmovapd Vpd, Wpd	vmovapd Wpd,Vpd	cvtpi2pd Vpd, Qpi	vmovntpd Mpd, Vpd	cvttpd2pi Ppi, Wpd	cvtpd2pi Qpi, Wpd	vucomisd Vsd, Wsd	vcomisd Vsd, Wsd
	F3			vcvtsi2ss Vss, Hss, Ey		vcvttss2si Gy, Wss	vcvtss2si Gy, Wss		
	F2			vcvtsi2sd Vsd, Hsd, Ey		vcvttsd2si Gy, Wsd	vcvtsd2si Gy, Wsd		
3	3	3-byte escape (Table A-4)		3-byte escape (Table A-5)					
4	4	CMOVcc(Gv, Ev) - Conditional Move S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
5		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, Hx, Wss	vcvttps2dq 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	vcvtsd2ss Vss, Hx, 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 Vdq, Hdq, Wdq	vpunpckhwd Vdq, Hdq, Wdq	vpunpckhdq Vdq, Hdq, Wdq	vpackssdw Vdq, Hdq, Wdq	vpunpcklqdq Vdq, Hdq, Wdq	vpunpckhqdq Vdq, Hdq, Wdq	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	vhsubpd Vpd, Hpd, Wpd	vmovd/q Ey, Vy	vmovdqa Wx,Vx
	F3							vmovq Vq, Wq	vmovdqu Wx,Vx
	F2					vhaddps Vps, Hps, Wps	vhsubps Vps, Hps, Wps		

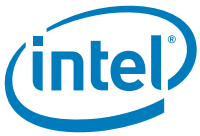


Table A-3. Two-byte Opcode Map: 80H — F7H (First Byte is 0FH) *

	pfx	0	1	2	3	4	5	6	7
8		Jcc[f64], Jz - Long-displacement jump on condition O	NO	B/CNAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
9		SETcc, Eb - Byte Set on condition O	NO	B/C/NAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
A		PUSH[d64] FS	POP[d64] FS	CPUID	BT Ev, Gv	SHLD Ev, Gv, Ib	SHLD Ev, Gv, CL		
B		CMPXCHG Eb, Gb	CMPXCHG Ev, Gv	LSS Gv, Mp	BTR Ev, Gv	LFS Gv, Mp	LGS Gv, Mp	MOVZX Gv, Eb	MOVZX 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 Vdq, Hdq, Wdq	vpsrld Vdq, Hdq, Wdq	vpsrlq Vdq, Hdq, Wdq	vpaddq Vdq, Hdq, Wdq	vpmullw Vdq, Hdq, Wdq	vmovq Wq, Vq	vpmovmskb Gd, Udq
	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	pmulhuw Pq, Qq	pmulhw Pq, Qq		movntq Mq, Pq
	66	vpavgb Vdq, Hdq, Wdq	vpsraw Vdq, Hdq, Wdq	vpsrad Vdq, Hdq, Wdq	vpavgw Vdq, Hdq, Wdq	vpmulhuw Vdq, Hdq, Wdq	vpmulhw Vdq, Hdq, Wdq	vcvttpd2dq Vx, Wpd	vmovntdq Mx, Vx
	F3							vcvtdq2pd Vx, Wpd	
	F2							vcvtpd2dq 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 Vdq, Hdq, Wdq	vpslld Vdq, Hdq, Wdq	vpsllq Vdq, Hdq, Wdq	vpmuludq Vdq, Hdq, Wdq	vpmaddwd Vdq, Hdq, Wdq	vpsadbw Vdq, Hdq, Wdq	vmaskmovdqu Vdq, Udq
	F2	vlddqu Vx, Mx							

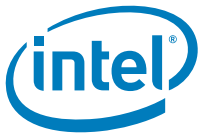


Table A-3. Two-byte Opcode Map: 88H — FFH (First Byte is 0FH) *

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

NOTES:

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

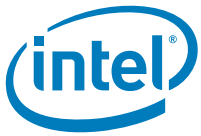


Table A-4 Three-byte Opcode Map: 00H — F7H (First Two Bytes are 0F 38H) *

	pfx	0	1	2	3	4	5	6	7
0		pshufb Pq, Qq	phaddw Pq, Qq	phaddd Pq, Qq	phaddsw Pq, Qq	pmaddubsw Pq, Qq	phsubw Pq, Qq	phsubd Pq, Qq	phsubsw Pq, Qq
	66	vpshufb Vdq, Hdq, Wdq	vphaddw Vdq, Hdq, Wdq	vphaddd Vdq, Hdq, Wdq	vphaddsw Vdq, Hdq, Wdq	vpmaddubsw Vdq, Hdq, Wdq	vphsubw Vdq, Hdq, Wdq	vphsubd Vdq, Hdq, Wdq	vphsubsw Vdq, Hdq, Wdq
1	66	pblendvb Vdq, Wdq			vcvtph2ps[V] Vx, Wx, Ib	blendvps Vdq, Wdq	blendvpd Vdq, Wdq		vptest Vx, Wx
2	66	vpmovsxbw Vdq, Udq/Mq	vpmovsxbd Vdq, Udq/Md	vpmovsxbq Vdq, Udq/Mw	vpmovsxwd Vdq, Udq/Mq	vpmovsxwq Vdq, Udq/Md	vpmovsxdq Vdq, Udq/Mq		
3	66	vpmovzxbw Vdq, Udq/Mq	vpmovzxbd Vdq, Udq/Md	vpmovzxbq Vdq, Udq/Mw	vpmovzxwd Vdq, Udq/Mq	vpmovzxwq Vdq, Udq/Md	vpmovzxdq Vdq, Udq/Mq		vpcmpgtq Vdq, Hdq, Wdq
4	66	vpmulld Vdq, Hdq, Wdq	vphminposuw Vdq, Wdq						
5									
6									
7									
8	66	INVEPT Gy, Mdq	INVVPID Gy, Mdq						
9									
A									
B									
C									
D									
E									
F		MOVBE Gy, My	MOVBE My, Gy						
	66	MOVBE Gw, Mw	MOVBE Mw, Gw						
	F3								
	F2	CRC32 Gd, Eb	CRC32 Gd, Ey						
	66 & F2	CRC32 Gd, Eb	CRC32 Gd, Ew						

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

	pfx	8	9	A	B	C	D	E	F
0		psignb Pq, Qq	psignw Pq, Qq	psignd Pq, Qq	pmulhrsw Pq, Qq				
	66	vpsignb Vdq, Hdq, Wdq	vpsignw Vdq, Hdq, Wdq	vpsignd Vdq, Hdq, Wdq	vpmulhrsw Vdq, Hdq, Wdq	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, Md	vbroadcastsd[V] Vqq, Mq	vbroadcastf128 [V] Vqq, Mdq		vpabsb Vdq, Wdq	vpabsw Vdq, Wdq	vpabsd Vdq, Wdq	
2	66	vpmuldq Vdq, Hdq, Wdq	vpcmpeqq Vdq, Hdq, Wdq	vmovntdqa Vdq, Mdq	vpackusdw Vdq, Hdq, Wdq	vmaskmovps[V] Vx,Hx,Mx	vmaskmovpd[V] Vx,Hx,Mx	vmaskmovps[V] Mx,Hx,Vx	vmaskmovpd[V] Mx,Hx,Vx
3	66	vpminsb Vdq, Hdq, Wdq	vpminsd Vdq, Hdq, Wdq	vpminuw Vdq, Hdq, Wdq	vpminud Vdq, Hdq, Wdq	vpmaxsb Vdq, Hdq, Wdq	vpmaxsd Vdq, Hdq, Wdq	vpmaxuw Vdq, Hdq, Wdq	vpmaxud Vdq, Hdq, Wdq
4									
5									
6									
7									
8									
9									
A									
B									
C									
D	66				VAESIMC Vdq, Wdq	VAESENC Vdq,Hdq,Wdq	VAESENCLAST Vdq,Hdq,Wdq	VAESDEC Vdq,Hdq,Wdq	VAESDECLAST Vdq,Hdq,Wdq
E									
F									
	66								
	F3								
	F2								
	66 & F2								

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

	pfx	0	1	2	3	4	5	6	7
0	66					vpermilps[V] Vx, Wx, Ib	vpermilpd[V] Vx, Wx, Ib	vperm2f128[V] Vqq,Hqq,Wqq,Ib	
1	66					vpextrb Rd/Mb, Vdq, Ib	vpextrw Rd/Mw, Vdq, Ib	vpextrd/q Ey, Vdq, Ib	vextractps Ed, Vdq, Ib
2	66	vpinsrb Vdq,Hdq, Ry/ Mb,Ib	vinsertps Vdq,Hdq, Udq/ Md,Ib	vpinsrd/q Vdq,Hdq,Ey,Ib					
3									
4	66	vdpps Vx,Hx,Wx,Ib	vdppd Vdq,Hdq,Wdq,Ib	vmpsadbw Vdq,Hdq,Wdq,Ib		vpclmulqdq Vdq,Hdq,Wdq,Ib			
5									
6	66	vpcmpestrm dq, Wdq, Ib	vpcmpestri Vdq, Wdq, Ib	vpcmpistrm Vdq, Wdq, Ib	vpcmpistri Vdq, Wdq, Ib				
7									
8									
9									
A									
B									
C									
D									
E									
F									

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

	pfx	8	9	A	B	C	D	E	F
0									palignr Pq, Qq, Ib
	66	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 Vdq,Hdq,Wdq,Ib	vpalignr Vdq,Hdq,Wdq,Ib
1	66	vinsertf128^{V} Vqq,Hqq,Wqq,Ib	vextractf128^{V} Wdq,Vqq,Ib				vcvtps2ph^{V} Wx, Vx, Ib		
2									
3									
4	66			vblendvps^{V} Vx,Hx,Wx,Lx	vblendvpd^{V} Vx,Hx,Wx,Lx	vpblendvb^{V} Vdq,Hdq,Wdq, Ldq			
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.

A.4 OPCODE EXTENSIONS FOR ONE-BYTE AND TWO-BYTE OPCODES

Some 1-byte and 2-byte opcodes use bits 3-5 of the ModR/M byte (the nnn field in Figure A-1) as an extension of the opcode.

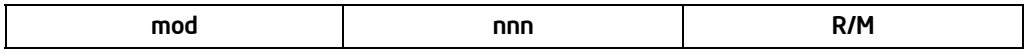


Figure A-1 ModR/M Byte nnn Field (Bits 5, 4, and 3)

Opcodes that have opcode extensions are indicated in Table A-6 and organized by group number. Group numbers (from 1 to 16, second column) provide a table entry point. The encoding for the r/m field for each instruction can be established using the third column of the table.

A.4.1 Opcode Look-up Examples Using Opcode Extensions

An Example is provided below.

Example A-4 Interpreting an ADD Instruction

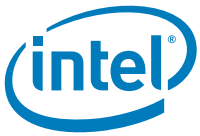
An ADD instruction with a 1-byte opcode of 80H is a Group 1 instruction:

- Table A-6 indicates that the opcode extension field encoded in the ModR/M byte for this instruction is 000B.
- The r/m field can be encoded to access a register (11B) or a memory address using a specified addressing mode (for example: mem = 00B, 01B, 10B).

Example A-5 Looking Up 0F01C3H

Look up opcode 0F01C3 for a VMRESUME instruction by using Table A-2, Table A-3 and Table A-6:

- 0F tells us that this instruction is in the 2-byte opcode map.
- 01 (row 0, column 1 in Table A-3) reveals that this opcode is in Group 7 of Table A-6.
- C3 is the ModR/M byte. The first two bits of C3 are 11B. This tells us to look at the second of the Group 7 rows in Table A-6.
- The Op/Reg bits [5,4,3] are 000B. This tells us to look in the 000 column for Group 7.
- Finally, the R/M bits [2,1,0] are 011B. This identifies the opcode as the VMRESUME instruction.



A.4.2 Opcode Extension Tables

See Table A-6 below.

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

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

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

Opcode	Group	Mod 7,6	pfx	Encoding of Bits 5,4,3 of the ModR/M Byte (bits 2,1,0 in parenthesis)							
				000	001	010	011	100	101	110	111
0F 71	12	mem									
		11B				psrlw Nq, Ib		psraw Nq, Ib		psllw Nq, Ib	
			66			vpsrlw Hdq,Udq,Ib		vpsraw Hdq,Udq,Ib		vpsllw Hdq,Udq,Ib	
0F 72	13	mem									
		11B				psrld Nq, Ib		psrad Nq, Ib		pslld Nq, Ib	
			66			vpsrld Hdq,Udq,Ib		vpsrad Hdq,Udq,Ib		vpslld Hdq,Udq,Ib	
0F 73	14	mem									
		11B				psrlq Nq, Ib				psllq Nq, Ib	
			66			vpsrlq Hdq,Udq,Ib	vpsrldq Hdq,Udq,Ib			vpsllq Hdq,Udq,Ib	vpslldq Hdq,Udq,Ib
0F AE	15	mem		fxsave	fxrstor	ldmxcsr	stmxcsr	XSAVE	XRSTOR	XSAVEOPT	clflush
		11B							lfence	mfence	sfence
			F3	RDFSBASE Ry	RDGSBASE Ry	WRFSBASE Ry	WRGSBASE Ry				
0F 18	16	mem		prefetch NTA	prefetch T0	prefetch T1	prefetch T2				
		11B									

NOTES:

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

...

6. Updates to Chapter 2, Volume 3A

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

...

2.5 CONTROL REGISTERS

...

OSXSAVE

XSAVE and Processor Extended States-Enable Bit (bit 18 of CR4) — When set, this flag: (1) indicates (via CPUID.01H:ECX.OSXSAVE[bit 27]) that the operating system supports the use of the XGETBV, XSAVE and XRSTOR instructions by general software; (2) enables the XSAVE and XRSTOR instructions to save and restore the x87 FPU state (including MMX registers), the SSE state (XMM registers and MXCSR), along with other processor extended states enabled in XCR0; (3) enables the processor to execute XGETBV and XSETBV instructions in order to read and write XCR0. See Section 2.6 and Chapter 13,

"System Programming for Instruction Set Extensions and Processor Extended States".

SMEP

SMEP-Enable Bit (bit 20 of CR4) — Enables supervisor-mode execution prevention (SMEP) when set. See Section 4.6, "Access Rights".

TPL

Task Priority Level (bit 3:0 of CR8) — This sets the threshold value corresponding to the highest-priority interrupt to be blocked. A value of 0 means all interrupts are enabled. This field is available in 64-bit mode. A value of 15 means all interrupts will be disabled.

...

7. Updates to Chapter 4, Volume 3A

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

...

4.1 PAGING MODES AND CONTROL BITS

Paging behavior is controlled by the following control bits:

- The WP and PG flags in control register CR0 (bit 16 and bit 31, respectively).
- The PSE, PAE, PGE, PCIDE, and SMEP flags in control register CR4 (bit 4, bit 5, bit 7, bit 17, and bit 20 respectively).
- The LME and NXE flags in the IA32_EFER MSR (bit 8 and bit 11, respectively).

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

Section 4.1.1 describes how the values of CR0.PG, CR4.PAE, and IA32_EFER.LME determine whether paging is in use and, if so, which of three paging modes is in use. Section 4.1.2 explains how to manage these bits to establish or make changes in paging modes. Section 0.0.1 discusses how CR0.WP, CR4.PSE, CR4.PGE, CR4.PCIDE, CR4.SMEP, and IA32_EFER.NXE modify the operation of the different paging modes.

4.1.1 Three Paging Modes

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

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

- If CR0.PG = 1 and CR4.PAE = 0, **32-bit paging** is used. 32-bit paging is detailed in Section 4.3. 32-bit paging uses CR0.WP, CR4.PSE, CR4.PGE, and CR4.SMEP as described in Section 0.0.1.
- If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0, **PAE paging** is used. PAE paging is detailed in Section 4.4. PAE paging uses CR0.WP, CR4.PGE, CR4.SMEP, and IA32_EFER.NXE as described in Section 0.0.1.
- If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 1, **IA-32e paging** is used.[1] IA-32e paging is detailed in Section 4.5. IA-32e paging uses CR0.WP, CR4.PGE, CR4.PCIDE, CR4.SMEP, and IA32_EFER.NXE as described in Section 0.0.1. IA-32e paging is available only on processors that support the Intel 64 architecture.

The three paging modes differ with regard to the following details:

- Linear-address width. The size of the linear addresses that can be translated.
- Physical-address width. The size of the physical addresses produced by paging.
- Page size. The granularity at which linear addresses are translated. Linear addresses on the same page are translated to corresponding physical addresses on the same page.
- Support for execute-disable access rights. In some paging modes, software can be prevented from fetching instructions from pages that are otherwise readable.
- Support for PCIDs. In some paging modes, software can enable a facility by which a logical processor caches information for multiple linear-address spaces. The processor may retain cached information when software switches between different linear-address spaces.

Table 4-1 illustrates the key differences between the three paging modes.

Table 4-1 Properties of Different Paging Modes

Paging Mode	PG in CR0	PAE in CR4	LME in IA32_EFER	Lin.-Addr. Width	Phys.-Addr. Width[1]	Page Sizes	Supports Execute-Disable?	Supports PCIDs?
None	0	N/A	N/A	32	32	N/A	No	No
32-bit	1	0	0[2]	32	Up to 40[3]	4 KB 4 MB[4]	No	No
PAE	1	1	0	32	Up to 52	4 KB 2 MB	Yes[5]	No
IA-32e	1	1	2	48	Up to 52	4 KB 2 MB 1 GB[6]	Yes[5]	Yes[7]

NOTES:

1. The physical-address width is always bounded by MAXPHYADDR; see Section 4.1.3.
2. The processor ensures that IA32_EFER.LME must be 0 if CR0.PG = 1 and CR4.PAE = 0.
3. 32-bit paging supports physical-address widths of more than 32 bits only for 4-MByte pages and only if the PSE-36 mechanism is supported; see Section 4.1.3 and Section 4.3.

---

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

4. 4-MByte pages are used with 32-bit paging only if CR4.PSE = 1; see Section 4.3.
5. Execute-disable access rights are applied only if IA32_EFER.NXE = 1; see Section 4.6.
6. Not all processors that support IA-32e paging support 1-GByte pages; see Section 4.1.3.
7. PCIDs are used only if CR4.PCIDE = 1; see Section 4.10.1.

Because they are used only if IA32_EFER.LME = 0, 32-bit paging and PAE paging is used only in legacy protected mode. Because legacy protected mode cannot produce linear addresses larger than 32 bits, 32-bit paging and PAE paging translate 32-bit linear addresses.

Because it is used only if IA32_EFER.LME = 1, IA-32e paging is used only in IA-32e mode. (In fact, it is the use of IA-32e paging that defines IA-32e mode.) IA-32e mode has two sub-modes:

- Compatibility mode. This mode uses only 32-bit linear addresses. IA-32e paging treats bits 47:32 of such an address as all 0.
- 64-bit mode. While this mode produces 64-bit linear addresses, the processor ensures that bits 63:47 of such an address are identical.[1] IA-32e paging does not use bits 63:48 of such addresses.

...

4.1.3 Paging-Mode Modifiers

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

- The WP flag in CR0 (bit 16).
- The PSE, PGE, PCIDE, and SMEP flags in CR4 (bit 4, bit 7, bit 17, and bit 20, respectively).
- The NXE flag in the IA32_EFER MSR (bit 11).

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

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

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

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

CR4.SMEP allows pages to be protected from supervisor-mode instruction fetches. If CR4.SMEP = 1, software operating with CPL < 3 (supervisor mode) cannot fetch instruc-

1. Such an address is called **canonical**. Use of a non-canonical linear address in 64-bit mode produces a general-protection exception (#GP(0)); the processor does not attempt to translate non-canonical linear addresses using IA-32e paging.

tions from linear addresses that are accessible in user mode (CPL = 3). Section 4.6 explains how access rights are determined.

IA32_EFER.NXE enables execute-disable access rights for PAE paging and IA-32e paging. If IA32_EFER.NXE = 1, instructions fetches can be prevented from specified linear addresses (even if data reads from the addresses are allowed). Section 4.6 explains how access rights are determined. (IA32_EFER.NXE has no effect with 32-bit paging. Software that wants to use this feature to limit instruction fetches from readable pages must use either PAE paging or IA-32e paging.)

4.1.4 Enumeration of Paging Features by CPUID

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

- PSE: page-size extensions for 32-bit paging.
  If CPUID.01H:EDX.PSE [bit 3] = 1, CR4.PSE may be set to 1, enabling support for 4-MByte pages with 32-bit paging (see Section 4.3).
- PAE: physical-address extension.
  If CPUID.01H:EDX.PAE [bit 6] = 1, CR4.PAE may be set to 1, enabling PAE paging (this setting is also required for IA-32e paging).
- PGE: global-page support.
  If CPUID.01H:EDX.PGE [bit 13] = 1, CR4.PGE may be set to 1, enabling the global-page feature (see Section 4.10.2.4).
- PAT: page-attribute table.
  If CPUID.01H:EDX.PAT [bit 16] = 1, the 8-entry page-attribute table (PAT) is supported. When the PAT is supported, three bits in certain paging-structure entries select a memory type (used to determine type of caching used) from the PAT (see Section 4.9.2).
- PSE-36: page-size extensions with 40-bit physical-address extension.
  If CPUID.01H:EDX.PSE-36 [bit 17] = 1, the PSE-36 mechanism is supported, indicating that translations using 4-MByte pages with 32-bit paging may produce physical addresses with up to 40 bits (see Section 4.3).
- PCID: process-context identifiers.
  If CPUID.01H:ECX.PCID [bit 17] = 1, CR4.PCIDE may be set to 1, enabling process-context identifiers (see Section 4.10.1).
- SMEP: supervisor-mode execution prevention.
  If CPUID.(EAX=07H,ECX=0H):EBX.SMEP [bit 7] = 1, CR4.SMEP may be set to 1, enabling supervisor-mode execution prevention (see Section 4.6).
- NX: execute disable.
  If CPUID.80000001H:EDX.NX [bit 20] = 1, IA32_EFER.NXE may be set to 1, allowing PAE paging and IA-32e paging to disable execute access to selected pages (see Section 4.6). (Processors that do not support CPUID function 80000001H do not allow IA32_EFER.NXE to be set to 1.)
- Page1GB: 1-GByte pages.
  If CPUID.80000001H:EDX.Page1GB [bit 26] = 1, 1-GByte pages are supported with IA-32e paging (see Section 4.5).
- LM: IA-32e mode support.
  If CPUID.80000001H:EDX.LM [bit 29] = 1, IA32_EFER.LME may be set to 1, enabling IA-32e paging. (Processors that do not support CPUID function 80000001H do not allow IA32_EFER.LME to be set to 1.)

- CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. (For processors that do not support CPUID function 80000008H, the width is generally 36 if CPUID.01H:EDX.PAE [bit 6] = 1 and 32 otherwise.) This width is referred to as MAXPHYADDR. MAXPHYADDR is at most 52.
- CPUID.80000008H:EAX[15:8] reports the linear-address width supported by the processor. Generally, this value is 48 if CPUID.80000001H:EDX.LM [bit 29] = 1 and 32 otherwise. (Processors that do not support CPUID function 80000008H, support a linear-address width of 32.)

...

4.3 32-BIT PAGING

A logical processor uses 32-bit paging if CR0.PG = 1 and CR4.PAE = 0. 32-bit paging translates 32-bit linear addresses to 40-bit physical addresses.[1] Although 40 bits corresponds to 1 TByte, linear addresses are limited to 32 bits; at most 4 GBytes of linear-address space may be accessed at any given time.

32-bit paging uses a hierarchy of paging structures to produce a translation for a linear address. CR3 is used to locate the first paging-structure, the page directory. Table 4-3 illustrates how CR3 is used with 32-bit paging.

32-bit paging may map linear addresses to either 4-KByte pages or 4-MByte pages. Figure 4-2 illustrates the translation process when it uses a 4-KByte page; Figure 4-3 covers the case of a 4-MByte page. The following items describe the 32-bit paging process in more detail as well has how the page size is determined:

- A 4-KByte naturally aligned page directory is located at the physical address specified in bits 31:12 of CR3 (see Table 4-3). A page directory comprises 1024 32-bit entries (PDEs). A PDE is selected using the physical address defined as follows:
  - Bits 39:32 are all 0.
  - Bits 31:12 are from CR3.
  - Bits 11:2 are bits 31:22 of the linear address.
  - Bits 1:0 are 0.

Because a PDE is identified using bits 31:22 of the linear address, it controls access to a 4-Mbyte region of the linear-address space. Use of the PDE depends on CR.PSE and the PDE's PS flag (bit 7):

- If CR4.PSE = 1 and the PDE's PS flag is 1, the PDE maps a 4-MByte page (see Table 4-4). The final physical address is computed as follows:
  - Bits 39:32 are bits 20:13 of the PDE.
  - Bits 31:22 are bits 31:22 of the PDE.[2]

1. Bits in the range 39:32 are 0 in any physical address used by 32-bit paging except those used to map 4-MByte pages. If the processor does not support the PSE-36 mechanism, this is true also for physical addresses used to map 4-MByte pages. If the processor does support the PSE-36 mechanism and MAXPHYADDR < 40, bits in the range 39:MAXPHYADDR are 0 in any physical address used to map a 4-MByte page. (The corresponding bits are reserved in PDEs.) See Section 4.1.3 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.
2. The upper bits in the final physical address do not all come from corresponding positions in the PDE; the physical-address bits in the PDE are not all contiguous.

— Bits 21:0 are from the original linear address.

- If CR4.PSE = 0 or the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 31:12 of the PDE (see Table 4-5). A page table comprises 1024 32-bit entries (PTEs). A PTE is selected using the physical address defined as follows:
  - Bits 39:32 are all 0.
  - Bits 31:12 are from the PDE.
  - Bits 11:2 are bits 21:12 of the linear address.
  - Bits 1:0 are 0.
- Because a PTE is identified using bits 31:12 of the linear address, every PTE maps a 4-KByte page (see Table 4-6). The final physical address is computed as follows:
  - Bits 39:32 are all 0.
  - Bits 31:12 are from the PTE.
  - Bits 11:0 are from the original linear address.

If a paging-structure entry's P flag (bit 0) is 0 or if the entry sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. A reference using a linear address whose translation would use such a paging-structure entry causes a page-fault exception (see Section 4.7).

With 32-bit paging, there are reserved bits only if CR4.PSE = 1:

- If the P flag and the PS flag (bit 7) of a PDE are both 1, the bits reserved depend on MAXPHYADDR whether the PSE-36 mechanism is supported:[1]
  - If the PSE-36 mechanism is not supported, bits 21:13 are reserved.
  - If the PSE-36 mechanism is supported, bits 21:(M–19) are reserved, where M is the minimum of 40 and MAXPHYADDR.
- If the PAT is not supported:[2]
  - If the P flag of a PTE is 1, bit 7 is reserved.
  - If the P flag and the PS flag of a PDE are both 1, bit 12 is reserved.

(If CR4.PSE = 0, no bits are reserved with 32-bit paging.)

...

4.4.1 PDPTE Registers

When PAE paging is used, CR3 references the base of a 32-Byte **page-directory-pointer table**. Table 4-7 illustrates how CR3 is used with PAE paging.

Table 4-7 Use of CR3 with PAE Paging

Bit Position(s)	Contents
4:0	Ignored

1. See Section 4.1.3 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.
2. See Section 4.1.3 for how to determine whether the PAT is supported.

Table 4-7 Use of CR3 with PAE Paging (Contd.)

Bit Position(s)	Contents
31:5	Physical address of the 32-Byte aligned page-directory-pointer table used for linear-address translation
63:32	Ignored (these bits exist only on processors supporting the Intel-64 architecture)

The page-directory-pointer-table comprises four (4) 64-bit entries called PDPTEs. Each PDPTE controls access to a 1-GByte region of the linear-address space. Corresponding to the PDPTEs, the logical processor maintains a set of four (4) internal, non-architectural PDPTE registers, called PDPTE0, PDPTE1, PDPTE2, and PDPTE3. The logical processor loads these registers from the PDPTEs in memory as part of certain operations:

- If PAE paging would be in use following an execution of MOV to CR0 or MOV to CR4 (see Section 4.1.1) and the instruction is modifying any of CR0.CD, CR0.NW, CR0.PG, CR4.PAE, CR4.PGE, CR4.PSE, or CR4.SMEP; then the PDPTEs are loaded from the address in CR3.
- If MOV to CR3 is executed while the logical processor is using PAE paging, the PDPTEs are loaded from the address being loaded into CR3.
- If PAE paging is in use and a task switch changes the value of CR3, the PDPTEs are loaded from the address in the new CR3 value.
- Certain VMX transitions load the PDPTE registers. See Section 4.11.1.

...

4.6 ACCESS RIGHTS

There is a translation for a linear address if the processes described in Section 4.3, Section 4.4.2, and Section 4.5 (depending upon the paging mode) completes and produces a physical address. The accesses permitted by a translation is determined by the access rights specified by the paging-structure entries controlling the translation.[1] The following items detail how paging determines access rights:

- For accesses in supervisor mode (CPL < 3):
  - Data reads.
    Data may be read from any linear address with a valid translation.
  - Data writes.
    - If CR0.WP = 0, data may be written to any linear address with a valid translation.
    - If CR0.WP = 1, data may be written to any linear address with a valid translation for which the R/W flag (bit 1) is 1 in every paging-structure entry controlling the translation.
  - Instruction fetches.
    - For 32-bit paging or if IA32_EFER.NXE = 0, access rights depend on the value of CR4.SMEP:

1. With PAE paging, the PDPTEs do not determine access rights.

— If CR4.SMEP = 0, instructions may be fetched from any linear address with a valid translation.

— If CR4.SMEP = 1, instructions may be fetched from any linear address with a valid translation for which the U/S flag (bit 2) is 0 in at least one of the paging-structure entries controlling the translation.

- For PAE paging or IA-32e paging with IA32_EFER.NXE = 1, access rights depend on the value of CR4.SMEP:

— If CR4.SMEP = 0, instructions may be fetched from any linear address with a valid translation for which the XD flag (bit 63) is 0 in every paging-structure entry controlling the translation.

— If CR4.SMEP = 1, instructions may be fetched from any linear address with a valid translation for which (1) the U/S flag is 0 in at least one of the paging-structure entries controlling the translation; and (2) the XD flag is 0 in every paging-structure entry controlling the translation.

- For accesses in user mode (CPL = 3):

— Data reads.
Data may be read from any linear address with a valid translation for which the U/S flag (bit 2) is 1 in every paging-structure entry controlling the translation.

— Data writes.
Data may be written to any linear address with a valid translation for which both the R/W flag and the U/S flag are 1 in every paging-structure entry controlling the translation.

— Instruction fetches.

- For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any linear address with a valid translation for which the U/S flag is 1 in every paging-structure entry controlling the translation.
- For PAE paging or IA-32e paging with IA32_EFER.NXE = 1, instructions may be fetched from any linear address with a valid translation for which the U/S flag is 1 and the XD flag is 0 in every paging-structure entry controlling the translation.

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 4.10). These structures may include information about access rights. The processor may enforce access rights based on the TLBs and paging-structure caches instead of on the paging structures in memory.

This fact implies that, if software modifies a paging-structure entry to change access rights, the processor might not use that change for a subsequent access to an affected linear address (see Section 4.10.4.3). See Section 4.10.4.2 for how software can ensure that the processor uses the modified access rights.

4.7 PAGE-FAULT EXCEPTIONS

Accesses using linear addresses may cause **page-fault exceptions** (#PF; exception 14). An access to a linear address may cause page-fault exception for either of two reasons: (1) there is no valid translation for the linear address; or (2) there is a valid translation for the linear address, but its access rights do not permit the access.

As noted in Section 4.3, Section 4.4.2, and Section 4.5, there is no valid translation for a linear address if the translation process for that address would use a paging-structure

entry in which the P flag (bit 0) is 0 or one that sets a reserved bit. If there is a valid translation for a linear address, its access rights are determined as specified in Section 4.6.

Figure 4-12 illustrates the error code that the processor provides on delivery of a page-fault exception. The following items explain how the bits in the error code describe the nature of the page-fault exception:

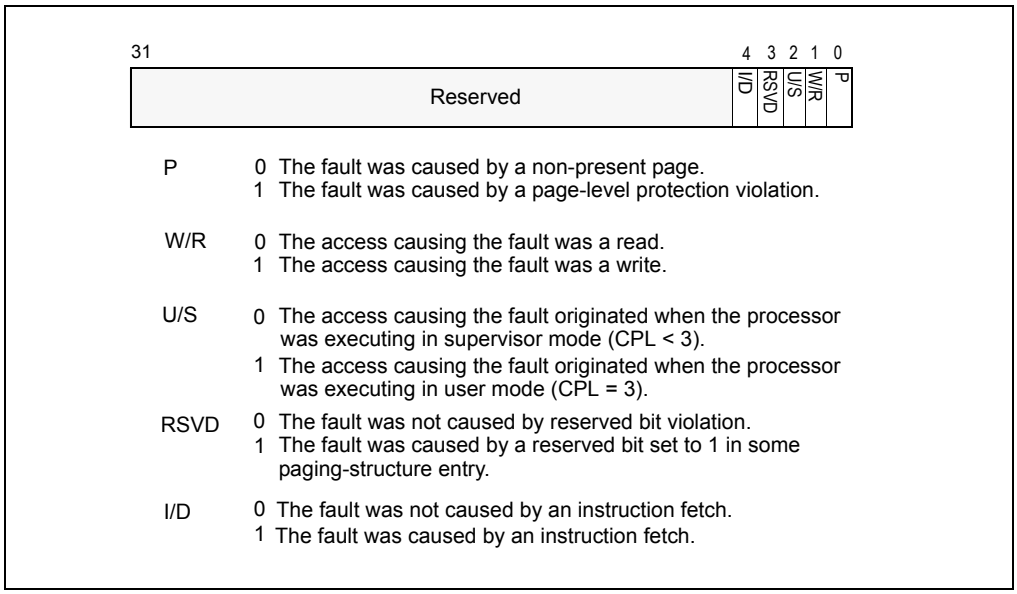


Figure 4-12 Page-Fault Error Code

- P flag (bit 0).
  This flag is 0 if there is no valid translation for the linear address because the P flag was 0 in one of the paging-structure entries used to translate that address.
- W/R (bit 1).
  If the access causing the page-fault exception was a write, this flag is 1; otherwise, it is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- U/S (bit 2).
  If a user-mode (CPL= 3) access caused the page-fault exception, this flag is 1; it is 0 if a supervisor-mode (CPL < 3) access did so. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- RSVD flag (bit 3).
  This flag is 1 if there is no valid translation for the linear address because a reserved bit was set in one of the paging-structure entries used to translate that address. (Because reserved bits are not checked in a paging-structure entry whose P flag is 0, bit 3 of the error code can be set only if bit 0 is also set.)

  Bits reserved in the paging-structure entries are reserved for future functionality. Software developers should be aware that such bits may be used in the future and that a paging-structure entry that causes a page-fault exception on one processor might not do so in the future.
- I/D flag (bit 4).
  This flag is 1 if (1) the access causing the page-fault exception was an instruction fetch; and (2) either (a) CR4.SMEP = 1; or (b) both (i) CR4.PAE = 1 (either PAE

paging or IA-32e paging is in use); and (ii) IA32_EFER.NXE = 1. Otherwise, the flag is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.

Page-fault exceptions occur only due to an attempt to use a linear address. Failures to load the PDPTE registers with PAE paging (see Section 4.4.1) cause general-protection exceptions (#GP(0)) and not page-fault exceptions.

...

4.10.4.1 Operations that Invalidate TLBs and Paging-Structure Caches

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

- INVLPG. This instruction takes a single operand, which is a linear address. The instruction invalidates any TLB entries that are for a page number corresponding to the linear address and that are associated with the current PCID. It also invalidates any global TLB entries with that page number, regardless of PCID (see Section 4.10.2.4).[1] INVLPG also invalidates all entries in all paging-structure caches associated with the current PCID, regardless of the linear addresses to which they correspond.
- MOV to CR0. The instruction invalidates all TLB entries (including global entries) and all entries in all paging-structure caches (for all PCIDs) if it changes the value of CR0.PG from 1 to 0.
- MOV to CR3. The behavior of the instruction depends on the value of CR4.PCIDE:
  - If CR4.PCIDE = 0, the instruction invalidates all TLB entries associated with PCID 000H except those for global pages. It also invalidates all entries in all paging-structure caches associated with PCID 000H.
  - If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 0, the instruction invalidates all TLB entries associated with the PCID specified in bits 11:0 of the instruction's source operand except those for global pages. It also invalidates all entries in all paging-structure caches associated with that PCID. It is not required to invalidate entries in the TLBs and paging-structure caches that are associated with other PCIDs.
  - If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 1, the instruction is not required to invalidate any TLB entries or entries in paging-structure caches.
- MOV to CR4. The behavior of the instruction depends on the bits being modified:
  - The instruction invalidates all TLB entries (including global entries) and all entries in all paging-structure caches (for all PCIDs) if (1) it changes the value of CR4.PGE;[2] or (2) it changes the value of the CR4.PCIDE from 1 to 0.
  - The instruction invalidates all TLB entries and all entries in all paging-structure caches for the current PCID if (1) it changes the value of CR4.PAE; or (2) it changes the value of CR4.SMEP from 0 to 1.

1. If the paging structures map the linear address using a page larger than 4 KBytes and there are multiple TLB entries for that page (see Section 4.10.2.3), the instruction invalidates all of them.
2. If CR4.PGE is changing from 0 to 1, there were no global TLB entries before the execution; if CR4.PGE is changing from 1 to 0, there will be no global TLB entries after the execution.

- Task switch. If a task switch changes the value of CR3, it invalidates all TLB entries associated with PCID 000H except those for global pages. It also invalidates all entries in all paging-structure caches for associated with PCID 000H.[1]
- VMX transitions. See Section 4.11.1.

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

- INVLPG may invalidate TLB entries for pages other than the one corresponding to its linear-address operand. It may invalidate TLB entries and paging-structure-cache entries associated with PCIDs other than the current PCID.
- MOV to CR0 may invalidate TLB entries even if CR0.PG is not changing. For example, this may occur if either CR0.CD or CR0.NW is modified.
- MOV to CR3 may invalidate TLB entries for global pages. If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 0, it may invalidate TLB entries and entries in the paging-structure caches associated with PCIDs other than the current PCID. It may invalidate entries if CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 1.
- MOV to CR4 may invalidate TLB entries when changing CR4.PSE or when changing CR4.SMEP from 1 to 0.
- On a processor supporting Hyper-Threading Technology, invalidations performed on one logical processor may invalidate entries in the TLBs and paging-structure caches used by other logical processors.

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

In addition to the instructions identified above, page faults invalidate entries in the TLBs and paging-structure caches. In particular, a page-fault exception resulting from an attempt to use a linear address will invalidate any TLB entries that are for a page number corresponding to that linear address and that are associated with the current PCID. it also invalidates all entries in the paging-structure caches that would be used for that linear address and that are associated with the current PCID.[2] These invalidations ensure that the page-fault exception will not recur (if the faulting instruction is re-executed) if it would not be caused by the contents of the paging structures in memory (and if, therefore, it resulted from cached entries that were not invalidated after the paging structures were modified in memory).

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

4.10.4.2 Recommended Invalidation

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

---

1. Task switches do not occur in IA-32e mode and thus cannot occur with IA-32e paging. Since CR4.PCIDE can be set only with IA-32e paging, task switches occur only with CR4.PCIDE = 0.
2. Unlike INVLPG, page faults need not invalidate **all** entries in the paging-structure caches, only those that would be used to translate the faulting linear address.

- If software modifies a paging-structure entry that identifies the final page frame for a page number (either a PTE or a paging-structure entry in which the PS flag is 1), it should execute INVLPG for any linear address with a page number whose translation uses that PTE.[1]

  (If the paging-structure entry may be used in the translation of different page numbers — see Section 4.10.3.3 — software should execute INVLPG for linear addresses with each of those page numbers; alternatively, it could use MOV to CR3 or MOV to CR4.)
- If software modifies a paging-structure entry that references another paging structure, it may use one of the following approaches depending upon the types and number of translations controlled by the modified entry:
  - — Execute INVLPG for linear addresses with each of the page numbers with translations that would use the entry. However, if no page numbers that would use the entry have translations (e.g., because the P flags are 0 in all entries in the paging structure referenced by the modified entry), it remains necessary to execute INVLPG at least once.
  - — Execute MOV to CR3 if the modified entry controls no global pages.
  - — Execute MOV to CR4 to modify CR4.PGE.
- If CR4.PCIDE = 1 and software modifies a paging-structure entry that does not map a page or in which the G flag (bit 8) is 0, additional steps are required if the entry may be used for PCIDs other than the current one. Any one of the following suffices:
  - — Execute MOV to CR4 to modify CR4.PGE, either immediately or before again using any of the affected PCIDs. For example, software could use different (previously unused) PCIDs for the processes that used the affected PCIDs.
  - — For each affected PCID, execute MOV to CR3 to make that PCID current (and to load the address of the appropriate PML4 table). If the modified entry controls no global pages and bit 63 of the source operand to MOV to CR3 was 0, no further steps are required. Otherwise, execute INVLPG for linear addresses with each of the page numbers with translations that would use the entry; if no page numbers that would use the entry have translations, execute INVLPG at least once.
- If software using PAE paging modifies a PDPTE, it should reload CR3 with the register's current value to ensure that the modified PDPTE is loaded into the corresponding PDPTE register (see Section 4.4.1).
- If the nature of the paging structures is such that a single entry may be used for multiple purposes (see Section 4.10.3.3), software should perform invalidations for all of these purposes. For example, if a single entry might serve as both a PDE and PTE, it may be necessary to execute INVLPG with two (or more) linear addresses, one that uses the entry as a PDE and one that uses it as a PTE. (Alternatively, software could use MOV to CR3 or MOV to CR4.)
- As noted in Section 4.10.2, the TLBs may subsequently contain multiple translations for the address range if software modifies the paging structures so that the page size used for a 4-KByte range of linear addresses changes. A reference to a linear address in the address range may use any of these translations.

  Software wishing to prevent this uncertainty should not write to a paging-structure entry in a way that would change, for any linear address, both the page size and either the page frame, access rights, or other attributes. It can instead use the following algorithm: first clear the P flag in the relevant paging-structure entry (e.g.,

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

PDE); then invalidate any translations for the affected linear addresses (see above); and then modify the relevant paging-structure entry to set the P flag and establish modified translation(s) for the new page size.

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

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

4.10.4.3 Optional Invalidation

The following items describe cases in which software may choose not to invalidate and the potential consequences of that choice:

- If a paging-structure entry is modified to change the P flag from 0 to 1, no invalidation is necessary. This is because no TLB entry or paging-structure cache entry is created with information from a paging-structure entry in which the P flag is 0.[1]
- If a paging-structure entry is modified to change the accessed flag from 0 to 1, no invalidation is necessary (assuming that an invalidation was performed the last time the accessed flag was changed from 1 to 0). This is because no TLB entry or paging-structure cache entry is created with information from a paging-structure entry in which the accessed flag is 0.
- If a paging-structure entry is modified to change the R/W flag from 0 to 1, failure to perform an invalidation may result in a "spurious" page-fault exception (e.g., in response to an attempted write access) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 4.10.4.1).
- If CR4.SMEP = 0 and a paging-structure entry is modified to change the U/S flag from 0 to 1, failure to perform an invalidation may result in a "spurious" page-fault exception (e.g., in response to an attempted user-mode access) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 4.10.4.1).
- If a paging-structure entry is modified to change the XD flag from 1 to 0, failure to perform an invalidation may result in a "spurious" page-fault exception (e.g., in response to an attempted instruction fetch) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 4.10.4.1).
- If a paging-structure entry is modified to change the accessed flag from 1 to 0, failure to perform an invalidation may result in the processor not setting that bit in response to a subsequent access to a linear address whose translation uses the entry. Software cannot interpret the bit being clear as an indication that such an access has not occurred.
- If software modifies a paging-structure entry that identifies the final physical address for a linear address (either a PTE or a paging-structure entry in which the PS flag is 1) to change the dirty flag from 1 to 0, failure to perform an invalidation may

1. If it is also the case that no invalidation was performed the last time the P flag was changed from 1 to 0, the processor may use a TLB entry or paging-structure cache entry that was created when the P flag had earlier been 1.

result in the processor not setting that bit in response to a subsequent write to a linear address whose translation uses the entry. Software cannot interpret the bit being clear as an indication that such a write has not occurred.

- The read of a paging-structure entry in translating an address being used to fetch an instruction may appear to execute before an earlier write to that paging-structure entry if there is no serializing instruction between the write and the instruction fetch. Note that the invalidating instructions identified in Section 4.10.4.1 are all serializing instructions.
- Section 4.10.3.3 describes situations in which a single paging-structure entry may contain information cached in multiple entries in the paging-structure caches. Because all entries in these caches are invalidated by any execution of INVLPG, it is not necessary to follow the modification of such a paging-structure entry by executing INVLPG multiple times solely for the purpose of invalidating these multiple cached entries. (It may be necessary to do so to invalidate multiple TLB entries.)

...

8. Updates to Chapter 8, Volume 3A

Change bars show changes to Chapter 8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

8.2.4 Out-of-Order Stores and Fast-String Operation

The Intel Core 2 Duo, Intel Core, Pentium 4, and P6 family processors modify the processors operation during the string store operations (initiated with the MOVS and STOS instructions) to maximize performance. This optimized operation (called **fast-string operation**) is used if certain initial conditions are met (see below). With fast-string operation, the processor operates on (from an external perspective) the string in a cache line by cache line mode. This results in the processor looping on issuing a cache-line read for the source address and an invalidation on the external bus for the destination address, knowing that all bytes in the destination cache line will be modified, for the length of the string. With fast-string operation, interrupts are accepted by the processor only on cache line boundaries. It is possible that, with fast-string operation, the destination line invalidations (and therefore stores) will be issued on the external bus out of order.

Code dependent upon sequential store ordering should not use string operations for the entire data structure to be stored. Data and semaphores should be separated. Order-dependent code should write to a discrete semaphore variable after any string operations to allow correctly ordered data to be seen by all processors.

Initial conditions for fast-string operation are implementation specific. Example conditions include:

- EDI and ESI must be 8-byte aligned for the Pentium III processor. EDI must be 8-byte aligned for the Pentium 4 processor.
- String operation must be performed in ascending address order.
- The initial operation counter (ECX) must be equal to or greater than 64.

- Source and destination must not overlap by less than a cache line (64 bytes, for Intel Core 2 Duo, Intel Core, Pentium M, and Pentium 4 processors; 32 bytes P6 family and Pentium processors).
- The memory type for both source and destination addresses must be either WB or WC.

NOTE

Initial conditions for fast-string operation in future Intel 64 or IA-32 processor families may differ from above.

Software can disable fast-string operation by clearing the fast-string-enable bit (bit 0) of IA32_MISC_ENABLE MSR. However, Intel recommends that system software always enable fast-string operation.

When fast-string operation is enabled (because IA32_MISC_ENABLE[0] = 1), some processors may further enhance the operation of the REP MOVSB and REP STOSB instructions. A processors supports these enhancements if CPUID.(EAX=07H, ECX=0H):EBX[bit 9] is 1.

...

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.

...

10.8.5 Signaling Interrupt Servicing Completion

For all interrupts except those delivered with the NMI, SMI, INIT, ExtINT, the start-up, or INIT-Deassert delivery mode, the interrupt handler must include a write to the end-of-interrupt (EOI) register (see Figure 10-21). This write must occur at the end of the handler routine, sometime before the IRET instruction. This action indicates that the servicing of the current interrupt is complete and the local APIC can issue the next interrupt from the ISR.

Figure 10-21 EOI Register

Upon receiving an EOI, the APIC clears the highest priority bit in the ISR and dispatches the next highest priority interrupt to the processor. If the terminated interrupt was a level-triggered interrupt, the local APIC also sends an end-of-interrupt message to all I/O APICs.

System software may prefer to direct EOIs to specific I/O APICs rather than having the local APIC send end-of-interrupt messages to all I/O APICs.

Software can inhibit the broadcast of EOI message by setting bit 12 of the Spurious Interrupt Vector Register (see Section 10.9). If this bit is set, a broadcast EOI is not generated on an EOI cycle even if the associated TMR bit indicates that the current interrupt was level-triggered. The default value for the bit is 0, indicating that EOI broadcasts are performed.

Bit 12 of the Spurious Interrupt Vector Register is reserved to 0 if the processor does not support suppression of EOI broadcasts. Support for EOI-broadcast suppression is reported in bit 24 in the Local APIC Version Register (see Section 10.4.8); the feature is supported if that bit is set to 1. When supported, the feature is available in both xAPIC mode and x2APIC mode.

System software desiring to perform directed EOIs for level-triggered interrupts should set bit 12 of the Spurious Interrupt Vector Register and follow each the EOI to the local xAPIC for a level triggered interrupt with a directed EOI to the I/O APIC generating the interrupt (this is done by writing to the I/O APIC's EOI register). System software performing directed EOIs must retain a mapping associating level-triggered interrupts with the I/O APICs in the system.

...

10.12.1.2 x2APIC Register Address Space

The MSR address range 800H through BFFH is architecturally reserved and dedicated for accessing APIC registers in x2APIC mode. Table 10-6 lists the APIC registers that are available in x2APIC mode. When appropriate, the table also gives the offset at which each register is available on the page referenced by IA32_APIC_BASE[35:12] in xAPIC mode.

There is a one-to-one mapping between the x2APIC MSRs and the legacy xAPIC register offsets with the following exceptions:

- The Destination Format Register (DFR): The DFR, supported at offset 0E0H in xAPIC mode, is not supported in x2APIC mode. There is no MSR with address 80EH.
- The Interrupt Command Register (ICR): The two 32-bit registers in xAPIC mode (at offsets 300H and 310H) are merged into a single 64-bit MSR in x2APIC mode (with MSR address 830H). There is no MSR with address 831H.
- The SELF IPI register. This register is available only in x2APIC mode at address 83FH. In xAPIC mode, there is no register defined at offset 3F0H.

Addresses in the range 800H–BFFH that are not listed in Table 10-6 (including 80EH and 831H) are reserved. Executions of RDMSR and WRMSR that attempt to access such addresses cause general-protection exceptions.

The MSR address space is compressed to allow for future growth. Every 32 bit register on a 128-bit boundary in the legacy MMIO space is mapped to a single MSR in the local x2APIC MSR address space. The upper 32-bits of all x2APIC MSRs (except for the ICR) are reserved.

...

10. Updates to Chapter 11, Volume 3A

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

...

11.11.1 MTRR Feature Identification

The availability of the MTRR feature is model-specific. Software can determine if MTRRs are supported on a processor by executing the CPUID instruction and reading the state of the MTRR flag (bit 12) in the feature information register (EDX).

If the MTRR flag is set (indicating that the processor implements MTRRs), additional information about MTRRs can be obtained from the 64-bit IA32_MTRRCAP MSR (named MTRRcap MSR for the P6 family processors). The IA32_MTRRCAP MSR is a read-only MSR that can be read with the RDMSR instruction. Figure 11-5 shows the contents of the IA32_MTRRCAP MSR. The functions of the flags and field in this register are as follows:

- **VCNT (variable range registers count) field, bits 0 through 7** — Indicates the number of variable ranges implemented on the processor.
- **FIX (fixed range registers supported) flag, bit 8** — Fixed range MTRRs (IA32_MTRR_FIX64K_00000 through IA32_MTRR_FIX4K_0F8000) are supported when set; no fixed range registers are supported when clear.
- **WC (write combining) flag, bit 10** — The write-combining (WC) memory type is supported when set; the WC type is not supported when clear.
- **SMRR (System-Management Range Register) flag, bit 11** — The system-management range register (SMRR) interface is supported when bit 11 is set; the SMRR interface is not supported when clear.

Bit 9 and bits 12 through 63 in the IA32_MTRRCAP MSR are reserved. If software attempts to write to the IA32_MTRRCAP MSR, a general-protection exception (#GP) is generated.

Software must read IA32_MTRRCAP VCNT field to determine the number of variable MTRRs and query other feature bits in IA32_MTRRCAP to determine additional capabilities that are supported in a processor. For example, some processors may report a value of '8' in the VCNT field, other processors may report VCNT with different values.

...

11. Updates to Chapter 16, Volume 3A

Change bars show changes to Chapter 16 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

16.4.4.1 Branch Trace Message Visibility

Branch trace message (BTM) visibility is implementation specific and limited to systems with a front side bus (FSB). BTMs may not be visible to newer system link interfaces or a system bus that deviates from a traditional FSB.

...

12. Updates to Chapter 28, Volume 3B

Change bars show changes to Chapter 28 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

28.3.5.2 Response to Page Faults

Page faults can occur for a variety of reasons. In some cases, the page fault alerts the VMM to an inconsistency between the active and guest page-table hierarchy. In such cases, the VMM can update the former and re-execute the faulting instruction. In other cases, the hierarchies are already consistent and the fault should be handled by the guest operating system. The VMM can detect this and use an established mechanism for raising a page fault to guest software.

The VMM can handle a page fault by following these steps (The steps below assume the guest is operating in a paging mode without PAE. Analogous steps to handle address translation using PAE or four-level paging mechanisms can be derived by VMM developers according to the paging behavior defined in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*):

1. First consult the active PDE, which can be located using the upper 10 bits of the faulting address and the current value of CR3. The active PDE is the source of the fault if it is marked not present or if its R/W bit and U/S bits are inconsistent with the attempted guest access (the guest privilege level and the values of CR0.WP and CR4.SMEP should also be taken into account).
2. If the active PDE is the source of the fault, consult the corresponding guest PDE using the same 10 bits from the faulting address and the physical address that corresponds to the guest address in the guest CR3. If the guest PDE would cause a page fault (for example: it is marked not present), then raise a page fault to the guest operating system.

   The following steps assume that the guest PDE would not have caused a page fault.
3. If the active PDE is the source of the fault and the guest PDE contains, as page-table base address (if PS = 0) or page base address (PS = 1), a guest address that the VMM has chosen not to support; then raise a machine check (or some other abort) to the guest operating system.

   The following steps assume that the guest address in the guest PDE is supported for the virtual machine.
4. If the active PDE is marked not-present, then set the active PDE to correspond to guest PDE as follows:
   a. If the active PDE contains a page-table base address (if PS = 0), then allocate an aligned 4-KByte active page table marked completely invalid and set the page-table base address in the active PDE to be the physical address of the newly allocated page table.
   b. If the active PDE contains a page base address (if PS = 1), then set the page base address in the active PDE to be the physical page base address that corresponds to the guest address in the guest PDE.
   c. Set the P, U/S, and PS bits in the active PDE to be identical to those in the guest PDE.

d. Set the PWT, PCD, and G bits according to the policy of the VMM.

e. Set A = 1 in the guest PDE.

f. If D = 1 in the guest PDE or PS = 0 (meaning that this PDE refers to a page table), then set the R/W bit in the active PDE as in the guest PDE.

g. If D = 0 in the guest PDE, PS = 1 (this is a 4-MByte page), and the attempted access is a write; then set R/W in the active PDE as in the guest PDE and set D = 1 in the guest PDE.

h. If D = 0 in the guest PDE, PS = 1, and the attempted access is not a write; then set R/W = 0 in the active PDE.

i. After modifying the active PDE, re-execute the faulting instruction.

The remaining steps assume that the active PDE is already marked present.

5. If the active PDE is the source of the fault, the active PDE refers to a 4-MByte page (PS = 1), the attempted access is a write; D = 0 in the guest PDE, and the active PDE has caused a fault solely because it has R/W = 0; then set R/W in the active PDE as in the guest PDE; set D = 1 in the guest PDE, and re-execute the faulting instruction.

6. If the active PDE is the source of the fault and none of the above cases apply, then raise a page fault of the guest operating system.

   The remaining steps assume that the source of the original page fault is not the active PDE.

NOTE

It is possible that the active PDE might be causing a fault even though the guest PDE would not. However, this can happen only if the guest operating system increased access in the guest PDE and did not take action to ensure that older translations were flushed from the TLB. Such translations might have caused a page fault if the guest software were running on bare hardware.

7. If the active PDE refers to a 4-MByte page (PS = 1) but is not the source of the fault, then the fault resulted from an inconsistency between the active page-table hierarchy and the processor's TLB. Since the transition to the VMM caused an address-space change and flushed the processor's TLB, the VMM can simply re-execute the faulting instruction.

   The remaining steps assume that PS = 0 in the active and guest PDEs.

8. Consult the active PTE, which can be located using the next 10 bits of the faulting address (bits 21–12) and the physical page-table base address in the active PDE. The active PTE is the source of the fault if it is marked not-present or if its R/W bit and U/S bits are inconsistent with the attempted guest access (the guest privilege level and the values of CR0.WP and CR4.SMEP should also be taken into account).

9. If the active PTE is not the source of the fault, then the fault has resulted from an inconsistency between the active page-table hierarchy and the processor's TLB. Since the transition to the VMM caused an address-space change and flushed the processor's TLB, the VMM simply re-executes the faulting instruction.

   The remaining steps assume that the active PTE is the source of the fault.

10. Consult the corresponding guest PTE using the same 10 bits from the faulting address and the physical address that correspond to the guest page-table base

address in the guest PDE. If the guest PTE would cause a page fault (it is marked not-present), the raise a page fault to the guest operating system.

The following steps assume that the guest PTE would not have caused a page fault.

11. If the guest PTE contains, as page base address, a physical address that is not valid for the virtual machine being supported; then raise a machine check (or some other abort) to the guest operating system.

    The following steps assume that the address in the guest PTE is valid for the virtual machine.

12. If the active PTE is marked not-present, then set the active PTE to correspond to guest PTE:

    a. Set the page base address in the active PTE to be the physical address that corresponds to the guest page base address in the guest PTE.

    b. Set the P, U/S, and PS bits in the active PTE to be identical to those in the guest PTE.

    c. Set the PWT, PCD, and G bits according to the policy of the VMM.

    d. Set A = 1 in the guest PTE.

    e. If D = 1 in the guest PTE, then set the R/W bit in the active PTE as in the guest PTE.

    f. If D = 0 in the guest PTE and the attempted access is a write, then set R/W in the active PTE as in the guest PTE and set D = 1 in the guest PTE.

    g. If D = 0 in the guest PTE and the attempted access is not a write, then set R/W = 0 in the active PTE.

    h. After modifying the active PTE, re-execute the faulting instruction.

    The remaining steps assume that the active PTE is already marked present.

13. If the attempted access is a write, D = 0 (not dirty) in the guest PTE and the active PTE has caused a fault solely because it has R/W = 0 (read-only); then set R/W in the active PTE as in the guest PTE, set D = 1 in the guest PTE and re-execute the faulting instruction.

14. If none of the above cases apply, then raise a page fault of the guest operating system.

...

13. Updates to Appendix A, Volume 3B

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

...

Table A-6. 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
...				
1EH	01H	INST_QUEUE_WRITE_CYCLES	This event counts the number of cycles during which instructions are written to the instruction queue. Dividing this counter by the number of instructions written to the instruction queue (INST_QUEUE_WRITES) yields the average number of instructions decoded each cycle. If this number is less than four and the pipe stalls, this indicates that the decoder is failing to decode enough instructions per cycle to sustain the 4-wide pipeline.	If SSE* instructions that are 6 bytes or longer arrive one after another, then front end throughput may limit execution speed.
...				
B0H	10H	OFFCORE_REQUESTS.ANY.RFO	Counts number of offcore RFO requests. Includes L2 prefetch requests.	
...				

...

14. Updates to Appendix B, Volume 3B

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

...

Model specific registers and its bit-fields may be supported for a finite range of processor families/models. To distinguish between different processor family and/or models, software must use CPUID.01H leaf function to query the combination of DisplayFamily and DisplayModel to determine model-specific availability of MSRs (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-M" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*). Table B-1 lists the signature values of DisplayFamily and DisplayModel for various processor families or processor number series.

Table B-1 CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_2DH	Next Generation Intel Xeon processor
06_2FH	Intel Xeon processor E7 family
06_2AH	Intel Xeon processor E3 family; Second Generation Intel Core i7, i5, i3 Processors 2xxx Series
06_2EH	Intel Xeon processor 7500, 6500 series
06_25H, 06_2CH	Intel Xeon processors 3600, 5600 series, Intel Core i7, i5 and i3 Processors
06_1EH, 06_1FH	Intel Core i7 and i5 Processors
06_1AH	Intel Core i7 Processor, Intel Xeon Processor 3400, 3500, 5500 series
06_1DH	Intel Xeon Processor MP 7400 series
06_17H	Intel Xeon Processor 3100, 3300, 5200, 5400 series, Intel Core 2 Quad processors 8000, 9000 series
06_0FH	Intel Xeon Processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad processor 6000 series, Intel Core 2 Extreme 6000 series, Intel Core 2 Duo 4000, 5000, 6000, 7000 series processors, Intel Pentium dual-core processors
06_0EH	Intel Core Duo, Intel Core Solo processors
06_0DH	Intel Pentium M processor
06_1CH	Intel Atom processor
0F_06H	Intel Xeon processor 7100, 5000 Series, Intel Xeon Processor MP, Intel Pentium 4, Pentium D processors
0F_03H, 0F_04H	Intel Xeon Processor, Intel Xeon Processor MP, Intel Pentium 4, Pentium D processors
06_09H	Intel Pentium M processor
0F_02H	Intel Xeon Processor, Intel Xeon Processor MP, Intel Pentium 4 processors
0F_0H, 0F_01H	Intel Xeon Processor, Intel Xeon Processor MP, Intel Pentium 4 processors
06_7H, 06_08H, 06_0AH, 06_0BH	Intel Pentium III Xeon Processor, Intel Pentium III Processor
06_03H, 06_05H	Intel Pentium II Xeon Processor, Intel Pentium II Processor
06_01H	Intel Pentium Pro Processor
05_01H, 05_02H, 05_04H	Intel Pentium Processor, Intel Pentium Processor with MMX Technology

…