# IA-32 Intel® Architecture Software Developer's Manual

## Documentation Changes

***February 2003***

**Notice:** The IA-32 Intel® Architecture 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 this specification update.

Document Number: 252046-003

# Contents

Revision History ..... 4
Preface ..... 5
Summary Table of Changes ..... 6
Documentation Changes ..... 7

# Revision History

| Version | Description | Date |
|---|---|---|
| -001 | Initial Release | November 2002 |
| -002 | Added 1-10 Documentation Changes.<br>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<br>Removed Documenation Change #6 - References to bits Gen and Len Deleted<br>Removed Documenation Change #4 - VIF Information Added to CLI Discussion | February 2003 |

# Preface

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

## Affected Documents/Related Documents

| Document Title | Document Number |
|---|---|
| *IA-32 Intel® Architecture Software Developer's Manual: Volume 1, Basic Architecture* | 245470-009 |
| *IA-32 Intel® Architecture Software Developer's Manual: Volume 2, Instruction Set Reference* | 245471-009 |
| *IA-32 Intel® Architecture Software Developer's Manual: Volume 3, System Programming Guide* | 245472-009 |

## Nomenclature

**Documentation Changes** include errors or omissions from the current published specifications. These changes will be incorporated in the next release of the Software Development Maunal.

# Summary Table of Changes

The following table indicates documentation changes which apply to the IA-32 Intel Architecture. This table uses the following notations:

## Codes Used in Summary Table

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

## Summary Table of Documentation Changes

| Number | DOCUMENTATION CHANGES |
|---|---|
| 1 | Description of FCOMI/FCOMIP Updated |
| 2 | Note Added to POP CS Instruction Entry in Table B-10 |
| 3 | Information Added on resource_stall and WC_Buffer |
| 4 | Switching the L1 Data Cache between Adaptive and Shared Mode |
| 5 | FTW Storage Location Corrected |
| 6 | More Information on Invalid TSS Conditions Provided |
| 7 | Memory Storage Location of x87 FPU Data Registers Specified for FSAVE |
| 8 | Valid Interrupt Vector Discussion Has More Explanation |
| 9 | Information Added to Clarify Handling of Reserved Bits |
| 10 | Various Corrections in Volume 2: Table B-23 |
| 11 | References to Bits GEn and Len Deleted |
| 12 | PSLLQ Pseudocode Error Corrected |
| 13 | STI and CLI Decision Tables Updated to Reflect VIF |
| 14 | Usage Note Added for Two Opcodes |
| 15 | #UD Exception Information Updated for Some Instructions |
| 16 | INT Pseudocode Updated with Flag Information |
| 17 | Errors Corrected in Opcode Table |

# Documentation Changes

All Documentation Changes will be incorporated into a future version of the IA-32 Intel® Architecture Software Developer's Manual.

## 1. Description of FCOMI/FCOMIP Updated

The *IA-32 Intel® Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, Chapter 3, Section: *FCOMI/FCOMIP/ FUCOMI/FUCOMIP-Compare Floating Point Values and Set EFLAGS* currently states:

The FUCOMI/FUCOMIP instructions always clear the OF flag in the EFLAGS register (regardless of whether an invalid-operation exception is detected); the FCOMI/FCOMIP instructions do not clear the OF flag.

It should state:

The FCOMI/FCOMIP and FUCOMI/FUCOMIP instructions clear the OF flag in the EFLAGS register (regardless of whether an invalid-operation exception is detected.

## 2. Note Added to POP CS Instruction Entry in Table B-10

The *IA-32 Intel® Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, Appendix B: *Instructions Formats and Encodings*, Table B-10: *General Purpose Instruction Formats and Encodings* has been updated by adding a note to the POP instruction. The note addresses a common error condition. The note is shown in the table segment below.

**Table B-10. General Purpose Instruction Formats and Encodings [table segment only, entire table not shown...]**

| | |
|---|---|
| **POP – Pop a Segment Register from the Stack**<br>(Note: CS cannot be sreg2 in this usage.) | |
| segment register DS, ES | 000 sreg2 111 |
| segment register SS | 000 sreg2 111 |
| segment register FS, GS | 0000 1111: 10 sreg3 001 |

## 3. Information Added on resource_stall and WC_Buffer

The *IA-32 Intel® Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Appendix A, Table 1 will now include the following information on resource_stall and WC_Buffer. The new data is shown in the table segments included below:

**Table A-1. Pentium 4 and Intel Xeon Processor Performance Monitoring Events for Non-Retirement Counting [table segment only, entire table not shown...]**

| Event Name | Parameters | Parameter Value | Description |
|---|---|---|---|
| resource_stall | | | This event monitors the occurrence or latency of stalls in the Allocator. |
| | ESCR restrictions | MSR_ALF_ESCR0<br>MSR_ALF_ESCR1 | |
| | Counter numbers per ESCR | ESCR0: 12, 13, 16<br>ESCR1: 14, 15, 17 | |
| | ESCR Event Select | 01H | ESCR[30:25] |
| | Event Masks | Bit<br>5: SBFULL | ESCR[24:9]<br>A Stall due to lack of store buffers |
| | CCCR Select | 01H | CCCR[15:13] |
| | Event Specific Notes | | This event may not be supported in all models of the processor family. |
| WC_Buffer | | | This event counts Write Combining Buffer operations that are selected by the event mask. |
| | ESCR restrictions | MSR_DAC_ESCR0<br>MSR_DAC_ESCR1 | |
| | Counter numbers per ESCR | ESCR0: 8, 9<br>ESCR1: 10, 11 | |
| | ESCR Event Select | 05H | ESCR[30:25] |
| | Event Masks | Bit<br>0: WCB_EVICTS<br>1: WCB_FULL _EVICT<br>2: WCB_HITM _EVICT | ESCR[24:9]<br>WC Buffer evictions of all causes<br>WC Buffer eviction: no WC buffer is available<br>WC Buffer eviction: Store encountered a Hit Modified condition |

## 4. Switching the L1 Data Cache between Adaptive and Shared Mode

The ability to switch the L1 Data Cache between adaptive and shared mode is now documented. In the *Intel Architecture Software Developer's Manual*, *Volume 2: Instruction Set Reference*, Chapter 3, Table 3-7; this capability impacts the description of ECX. See the EAX=1, ECX[10] in the following table segment:

**Table 3-7. Information Returned By CPUID Instruction [table segment only, entire table is not shown...]**

| Initial EAX Value | Information Provided About the Processor | |
|---|---|---|
| | | Basic CPUID Information |
| 1H | EAX | Version Information (Type, Family, Model, and Stepping ID) |
| | EBX | Bits 7-0: Brand Index<br>Bits 15-8: CLFLUSH line size. (Value ∗ 8 = cache line size in bytes)<br>Bits 23-16: Number of logical processors per physical processor.<br>Bits 31-24: Local APIC ID |
| | ECX | Bits 9-0: Reserved<br>**Bit 10: If this bit is 1, the L1 Data Cache may be placed in adaptive mode or shared mode. Mode selection is determined by Bit 24 of IA32_MISC_ENABLE (see Volume 3, Appendix B, Table B-1). If ECX[10] is 0, the ability to change the L1 Data Cache mode is not supported.**<br>Bits 63-11: Reserved |
| | EDX | Feature Information (see Figure 3-4 and Table 3-10) |

Supporting information has also been added to the *IA-32 Intel® Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Appendix B, Table B1. The description IA32_MISC_ENABLE[24] has been changed. This information has been provided in the table segment shown:

**Table B-1. Special Fields within Instruction Encodings [table segment only...]**

| Register Address | | Register Name Fields and Flags | Shared/ Unique[1] | Bit Description |
|---|---|---|---|---|
| Hex | Dec | | | |
| 1A0H | 416 | IA32_MISC_ENABLE | Shared | Enable Miscellaneous Processor Features. (R/W) Allows a variety of processor functions to be enabled and disabled. |
| | | 0 | | **Fast-Strings Enable.** When set, the fast-strings feature on the Pentium 4 processor is enabled(default); when clear, fast-strings are disabled. |
| | | 1 | | **Reserved.** |
| | | 2 | | **x87 FPU Fopcode Compatibility Mode Enable.** When set, fopcode compatibility mode is enabled; when clear (default), mode is disabled. See “Fopcode Compatibility Mode“ in Chapter 8 of the *IA-32 Intel Architecture Software Developer’s Manual, Volume 1*. |
| | | 3 | | **Thermal Monitor Enable.** When set, clock modulation controlled by the processor’s internal thermal sensor is enabled; when clear (default), automatic clock modulation is disabled. (See Section 13.14.2., “Automatic Thermal Monitor”.) |

| Table B-1. Special Fields within Instruction Encodings [table segment only...] | | | | |
|---|---|---|---|---|
| **Register Address** | | **Register Name Fields and Flags** | **Shared/ Unique**[1] | **Bit Description** |
| **Hex** | **Dec** | | | |
| | | 4 | | **Split-Lock Disable.** This debug feature is specific to the Pentium 4. When set, the bit causes an #AC exception to be issued instead of a split-lock cycle. Operating systems that set this bit must align system structures to avoid split-lock scenarios. When the bit is clear (the default), normal split-locks are issued to the bus. |
| | | 5 | | **Reserved.** |
| | | 6 | | **Third-Level Cache Disable.** (R/W) When set, the third-level cache is disabled; when clear (default) the third-level cache is enabled. This flag is reserved for processors that do not have a third-level cache. Note that this bit controls only the third-level cache, and then only if overall caching is enabled through the CD flag of control register CR0, the page-level cache controls, and/or the MTRRs (see Section 10.5.4., “Disabling and Enabling the L3 Cache”). |
| | | 7 | | **Performance Monitoring Available.** (R) When set, performance monitoring is enabled; when clear, performance monitoring is disabled. |
| | | 8 | | **Suppress Lock Enable.** When set assert on of lock on the bus is suppressed during a Split Lock access. when clear (default) does not suppress lock. |
| | | 9 | | **Prefetch Queue Disable.** When set disables the prefetch queue. When clear (default) the prefetch queue is enabled. |
| | | 10 | | **FERR# Interrupt Reporting Enable. (R/W)** When set, interrupt reporting through the FERR# pin is enabled; when clear, this interrupt reporting function is disabled. When this flag is set and the processor is in the stop-clock state (STPCLK# is asserted), asserting the FERR# pin signals to the processor that an interrupt (such as, INIT#, BINIT#, INTR, NMI, SMI#, or RESET#) is pending and that the processor should return to normal operation to handle the interrupt. This flag does not affect the normal operation of the FERR# pin (to indicate an unmasked floating-point error) when the STPCLK# pin is not asserted. |
| | | 11 | | **Branch Trace Storage Unavailable (BTS_UNAVILABLE).** (R) When set, the processor does not support branch trace storage (BTS); when clear, BTS is supported. |
| | | 12 | | **Precise Event Based Sampling Unavailable (PEBS_UNAVILABLE).** (R) When set, the processor does not support precise event-based sampling (PEBS); when clear, PEBS is supported. |

| Table B-1. Special Fields within Instruction Encodings [table segment only...] | | | | |
|---|---|---|---|---|
| Register Address | | Register Name Fields and Flags | Shared/ Unique[1] | Bit Description |
| Hex | Dec | | | |
| | | 23:13 | | **Reserved.** |
| | | 24 | | **L1 Data Cache Context Mode (R/W).** When set to 1, this bit places the L1 Data Cache into shared mode. When set to 0 (the default), this bit places the L1 Data Cache into adaptive mode. In adaptive mode, the Page Directory Base Register contained in CR3 must be identical across all logical processors.<br><br>Note: If the Context ID feature flag, ECX[10], is not set to 1 after executing CPUID with EAX = 1; the ability to switch modes is not supported and the BIOS must not alter the contents of IA32_MISC_ENABLE[24]. |
| | | 63:25 | | **Reserved.** |

## 5. FTW Storage Location Corrected

The *IA-32 Intel® Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, Chapter 3, Section: *FXSAVE - Save x87 FPU, MMX, SSE, and SSE2 State* incorrectly indicates the memory storage location of the x87 FPU Tag Word (FTW). The storage location has been corrected to indicate a byte 4 offset (instead of a byte 5 offset).

A partial representation of the corrected table is shown below.

**Table 3-14. Layout of FXSAVE and FXRSTOR Memory Region [table segment only, entire table is not shown...]**

| 15 14 | 13 12 | 11 10 | 9 8 | 7 6 | 5 | 4 | 3 2 | 1 0 | |
|---|---|---|---|---|---|---|---|---|---|
| Rsrvd | CS | FPU IP | | FOP | | **FTW** | FSW | FCW | **0** |
| MXCSR_MASK | | MXCSR | | Rsrvd | DS | | FPU DP | | **16** |
| Reserved | | ST0/MM0 | | | | | | | **32** |
| Reserved | | ST1/MM1 | | | | | | | **48** |
| Reserved | | ST2/MM2 | | | | | | | **64** |
| Reserved | | ST3/MM3 | | | | | | | **80** |
| Reserved | | ST4/MM4 | | | | | | | **96** |
| Reserved | | ST5/MM5 | | | | | | | **112** |
| Reserved | | ST6/MM6 | | | | | | | **128** |
| Reserved | | ST7/MM7 | | | | | | | **144** |
| XMM0 | | | | | | | | | **160** |

## 6. More Information on Invalid TSS Conditions Provided

In the *A-32 Intel® Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Chapter 5, Section: *Interrupt 10 - Invalid TSS Exception (#TS)*, Table 5-6 has been modified to include a more complete list of invalid conditions. The impacted part of the table is reproduced below with additions indicated by change bars.

**Table 5-6. Invalid TSS Conditions [table segment only, entire table not shown...]**

| Error Code Index | Invalid Condition |
|---|---|
| TSS segment selector index | TSS segment limit less than 67H for 32-bit TSS or less than 2CH for 16-bit TSS<br>During an IRET task switch, the TI flag in the TSS segment selector indicates the LDT<br>During an IRET task switch, the TSS segment selector exceeds descriptor table limit<br>During an IRET task switch, the busy flag in the TSS descriptor indicates an inactive task |
| LDT segment selector index | Invalid LDT or LDT not present |

In the *IA-32 Intel® Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Chapter 5, Section: *Interrupt 13 – General Protection Exception (#GP)* includes a large bulleted list. The list has been corrected. The corrected portion of the list is shown below.

- Loading the CS register with a segment selector for a data segment or a null segment selector.
- Accessing memory using the DS, ES, FS, or GS register when it contains a null segment selector.
- Switching to a busy task during a call or jump to a TSS.
- Using a segment selector on a non-IRET task switch that points to a TSS descriptor in the current LDT. TSS descriptors can only reside in the GDT. This condition causes a #TS exception during an IRET task switch.
- Violating any of the privilege rules described in Chapter 4, *Protection*.
- Exceeding the instruction length limit of 15 bytes (this only can occur when redundant prefixes are placed before an instruction).

## 7. Memory Storage Location of x87 FPU Data Registers Specified for FSAVE

In the *IA-32 Intel® Architecture Software Developer's Manual, Volume 1: Basic Architecture*, Chapter 8, Figures 8-9 and 8-10 have been updated to include a short note about storage of the X87 data registers during the operation of FSAVE. Both updated figures are shown below:

Figure 8-9. Protected Mode x87 FPU State Image in Memory, 32-Bit Format

**32-Bit Protected Mode Format**

| 31 ... 16 | 15 ... 0 | |
|---|---|---|
| | Control Word | 0 |
| | Status Word | 4 |
| | Tag Word | 8 |
| FPU Instruction Pointer Offset | | 12 |
| 0 0 0 0 0 Opcode 10...00 | FPU Instruction Pointer Selector | 16 |
| FPU Operand Pointer Offset | | 20 |
| | FPU Operand Pointer Selector | 24 |

For instructions that also store x87 FPU data registers, the eight 80-bit registers (R0-R7) follow the above structure in sequence.

Figure 8-10. Real Mode x87 FPU State Image in Memory, 32-Bit Format

**32-Bit Real-Address Mode Format**

| 31 ... 16 | 15 ... 0 | | |
|---|---|---|---|
| | Control Word | | 0 |
| | Status Word | | 4 |
| | Tag Word | | 8 |
| | FPU Instruction Pointer 15...00 | | 12 |
| 0 0 0 0 FPU Instruction Pointer 31...16 | 0 | Opcode 10...00 | 16 |
| | FPU Operand Pointer 15...00 | | 20 |
| 0 0 0 0 FPU Operand Pointer 31...16 | 0 0 0 0 0 0 0 0 0 0 0 0 | | 24 |

For instructions that also store x87 FPU data registers, the eight 80-bit registers (R0-R7) follow the above structure in sequence.

## 8. Valid Interrupt Vector Discussion Has More Explanation

In the *IA-32 Intel® Architecture Software Developer's Manual, Volume. 3: System Programming Guide*, Chapter 8, Section 8.5.2 has been updated to include more information. See the data below. Change bars note the updated data.

**Section 8.5.2 Valid Interrupt Vectors**

The IA-32 architecture defines 256 vector numbers, ranging from 0 through 255 (see Section 5.2., *Exception and Interrupt Vectors)*. The local and I/O APICs support 240 of these vectors (in the range of 16 to 255) as valid interrupts.

When an interrupt vector in the range of 0 to 15 is sent or received through the local APIC, the APIC indicates an illegal vector in its Error Status Register [see Section 8.5.3., *Error Handling*]. The IA-32 architecture reserves vectors 16 through 31 for predefined interrupts, exceptions, and Intel-reserved encodings (see Table 5-1); however, the local APIC does not treat vectors in this range as illegal.

When an illegal vector value (0 to 15) is written to an LVT entry and the delivery mode is Fixed (bits 8-11 equal 0), the APIC may signal an illegal vector error, without regard to whether the mask bit is set or whether an interrupt is actually seen on the input

## 9. Information Added to Clarify Handling of Reserved Bits

In the *Intel Architecture Software Developer's Manual*, Vol 2: *Instruction Set Reference*, Chapter 3, Section: *MOV—Move to/from Control Registers, Description* subsection; the second paragraph changed to document an error condition that can occur when loading control registers. Note the change bar and bold text:

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

## 10. Various Corrections in Volume 2: Table B-23

In the *Intel Architecture Software Developer's Manual*, Vol 2: *Instruction Set Reference*, Chapter 3, Appendix B, Table B-23: *Floating-Point Structures and Encodings*; various corrections have been made. Cells impacted by corrections are listed:

| [table segment only, entire table is not shown] | |
|---|---|
| **Instruction and Format** | **Encoding** |
| **...** | **...** |
| **FDIV – Divide** | |
| ST(0) ← ST(0) ÷ 32-bit memory | 11011 000 : mod 110 r/m |
| ST(0) ← ST(0) ÷ 64-bit memory | 11011 100 : mod 110 r/m |
| ST(d) ← ST(0) ÷ ST(i) | 11011 d00 : 1111 R ST(i) |
| **...** | **...** |
| **FDIVR – Reverse Divide** | |
| ST(0) ← 32-bit memory ÷ ST(0) | 11011 000 : mod 111 r/m |
| ST(0) ← 64-bit memory ÷ ST(0) | 11011 100 : mod 111 r/m |
| ST(d) ← ST(i) ÷ ST(0) | 11011 d00 : 1111 R ST(i) |
| **...** | **...** |
| **FIMUL** | |
| ST(0) ← ST(0) × 16-bit memory | 11011 110 : mod 001 r/m |
| ST(0) ← ST(0) × 32-bit memory | 11011 010 : mod 001 r/m |
| **...** | **...** |
| **FISUB** | |
| ST(0) ← ST(0) - 16-bit memory | 11011 110 : mod 100 r/m |
| ST(0) ← ST(0) - 32-bit memory | 11011 010 : mod 100 r/m |
| **FISUBR** | |
| ST(0) ← 16-bit memory – ST(0) | 11011 110 : mod 101 r/m |
| ST(0) ← 32-bit memory – ST(0) | 11011 010 : mod 101 r/m |
| **...** | **...** |
| **FMULP – Multiply** | |
| ST(i) ← ST(0) × ST(i) | 11011 110 : 1100 1 ST(i) |
| | |

## 11. References to Bits GEn and Len Deleted

The *Intel Architecture Software Developer's Manual,* Vol 3: *System Programming Guide,* Chapter 15, Table 15-2 *Debug Exception Condition* referenced bits Gen and Len. They don't exist in this context; this error has been removed. The corrected table is shown:

Table 15-2. Debug Exception Conditions

| Debug or Breakpoint Condition | DR6 Flags Tested | DR7 Flags Tested | Exception Class |
|---|---|---|---|
| Single-step trap | BS = 1 | | Trap |
| Instruction breakpoint, at addresses defined by DR*n* and LEN*n* | B*n* = 1 and (G*n* or L*n* = 1) | R/Wn = 0 | Fault |
| Data write breakpoint, at addresses defined by DR*n* and LEN*n* | B*n* = 1 and (G*n* or L*n* = 1) | R/Wn = 1 | Trap |
| I/O read or write breakpoint, at addresses defined by DR*n* and LEN*n* | B*n* = 1 and (G*n* or L*n* = 1) | R/Wn = 2 | Trap |
| Data read or write (but not instruction fetches), at addresses defined by DR*n* and LEN*n* | B*n* = 1 and (G*n* or L*n* = 1) | R/Wn = 3 | Trap |
| General detect fault, resulting from an attempt to modify debug registers (usually in conjunction with in-circuit emulation) | BD = 1 | | Fault |
| Task switch | BT = 1 | | Trap |

## 12. PSLLQ Pseudocode Error Corrected

In the *Intel Architecture Software Developer's Manual*, Vol 2: *Instruction Set Reference*, Chapter 3, Section: *PSLLW/PSLLD/PSLLQ—Shift Packed Data Left Logical*; we corrected a typo in the PSLLQ pseudocode. The corrected entry is shown below.

```
PSLLQ instruction with 128-bit operand:
    IF (COUNT > 63)
    THEN
        DEST[128..0] ← 00000000000000000000000000000000H
    ELSE
        DEST[63-0]  ← ZeroExtend(DEST[63-0] << COUNT);
        DEST[127-64] ← ZeroExtend(DEST[127-64] << COUNT);
    FI;
```

## 13. STI and CLI Decision Tables Updated to Reflect VIF

In the *Intel Architecture Software Developer's Manual*, Vol 2: *Instruction Set Reference*, Chapter 3, Section: *CLI — Clear Interrupt Flag*; we have added VIF information to the relevent decision table. The new CLI decision table is shown below:

| PE | VM | IOPL | CPL | PVI | VIP | VME | CLI Result |
|---|---|---|---|---|---|---|---|
| 0 | X | X | X | X | X | X | **IF = 0** |
| 1 | 0 | ≥ CPL | X | X | X | X | **IF = 0** |
| 1 | 0 | < CPL | 3 | 1 | X | X | **VIF = 0** |
| 1 | 0 | < CPL | < 3 | X | X | X | **GP Fault** |
| 1 | 0 | < CPL | X | 0 | X | X | **GP Fault** |
| 1 | 1 | 3 | X | X | X | X | **IF = 0** |
| 1 | 1 | < 3 | X | X | X | 1 | **VIF = 0** |
| 1 | 1 | < 3 | X | X | X | 0 | **GP Fault** |
| **X = This setting has no impact.** | | | | | | | |

In the *Intel Architecture Software Developer's Manual*, Vol 2: *Instruction Set Reference*, Chapter 3, Section: *STI — Set Interrupt Flag*; we have added VIF information to the relevent decision table. The new STI decision table is shown below:

| PE | VM | IOPL | CPL | PVI | VIP | VME | STI Result |
|---|---|---|---|---|---|---|---|
| 0 | X | X | X | X | X | X | **IF = 1** |
| 1 | 0 | ≥ CPL | X | X | X | X | **IF = 1** |
| 1 | 0 | < CPL | 3 | 1 | 0 | X | **VIF = 1** |
| 1 | 0 | < CPL | < 3 | X | X | X | **GP Fault** |
| 1 | 0 | < CPL | X | 0 | X | X | **GP Fault** |
| 1 | 0 | < CPL | X | X | 1 | X | **GP Fault** |
| 1 | 1 | 3 | X | X | X | X | **IF = 1** |
| 1 | 1 | < 3 | X | X | 0 | 1 | **VIF = 1** |
| 1 | 1 | < 3 | X | X | 1 | X | **GP Fault** |
| 1 | 1 | < 3 | X | X | X | 0 | **GP Fault** |
| **X = This setting has no impact.** | | | | | | | |

## 14. Usage Note Added for Two Opcodes

In the *Intel Architecture Software Developer's Manual*, Vol 2: *Instruction Set Reference*, Appendix A, Table A-3; we have added a usage note to two opcodes. Only impacted table cells are shown. Note the bold text.

| [table segment only, entire table not shown] | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | MOVUPS Vps, Wps<br>MOVSS (F3) Vss, Wss<br>MOVUPD (66) Vpd, Wpd<br>MOVSD (F2) Vsd, Wsd | MOVUPS Wps, Vps<br>MOVSS (F3) Wss, Vss<br>MOVUPD (66) Wpd, Vpd<br>MOVSD (F2) Vsd, Wsd | MOVLPS Wq, Vq<br>MOVLPD (66) Vq, Ws<br>MOVHLPS Vq, Vq | MOVLPS Vq, Wq<br>MOVLPD (66) Vq, Wq | UNPCKLPS Vps, Wq<br>UNPCKLPD (66) Vpd, Wq | UNPCKHPS Vps, Wq<br>UNPCKHPD (66) Vpd, Wq | MOVHPS Vq, Wq<br>MOVHPD (66) Vq, Wq<br>MOVLHPS Vq, Vq | MOVHPS Wq, Vq<br>MOVHPD (66) Wq, Vq |
| 2 | MOV Rd, Cd | MOV Rd, Dd | MOV Cd, Rd | MOV Dd, Rd | **MOV Rd, Td††** | | **MOV Td, Rd††** | |

**NOTE**:

† All blanks in the opcode map shown in Table A-3 are reserved and should not be used. Do not depend on the operation of these undefined opcodes.

†† Not currently supported after Pentium Pro and Pentium II families. Using this opcode on the current generation of processors will generate a #UD. For future processors, this value is reserved.

## 15. #UD Exception Information Updated for Some Instructions

In the *Intel Architecture Software Developer's Manual*, Vol 2: *Instruction Set Reference*, Chapter 3; we have updated #UD exception information in for the following instructions:

| | | | |
|---|---|---|---|
| MOVD xmm, r32 | MOVD r32, xmm | MOVDQA | MOVDQU |
| MOVQ xmm, m64 | PACKSSWB/DW | PACKUSWB | PADDB/W/D |
| PADDSB/W | PADDUSB/W | PAND | PANDN PCMPEQB/W/ |
| D | PCMPGTB/W/D | PMADDWD | PMULHWPMULLW |
| POR | PSLLW/D/Q | PSRAW/D | PSRLW/D/QPSUBB/W/ |
| D | PSUBSB/W | PSUBUSB/W | PUNPCKHBW/WD/DQ |
| PUNPCKLBW/WD/DQ | PXOR | | |

The updated #UD text is as follows.

### Protected Mode Exceptions

#UD — If EM in CR0 is set.

128-bit operations will generate #UD only if OSFXSR in CR4 is 0. Execution of 128-bit instructions on a non-SSE2 capable processor (one that is MMX capable) will result in the instruction operating on the mm registers, not #UD.

### Real-Address Mode Exceptions

#UD — If EM in CR0 is set.

128-bit operations will generate #UD only if OSFXSR in CR4 is 0. Execution of 128-bit instructions on a non-SSE2 capable processor (one that is MMX capable) will result in the instruction operating on the mm registers, not #UD.

## 16. INT Pseudocode Updated with Flag Information

In the *Intel Architecture Software Developer's Manual*, Vol 2: *Instruction Set Reference*, Section: *INT n/INTO/INT 3—Call to Interrupt Procedure*; we have updated the pseudocode to better reflect how some flags are cleared. The corrected listing is shown.

```
IF PE = 0
    THEN
        GOTO REAL-ADDRESS-MODE;
    ELSE (* PE = 1 *)
        IF (VM = 1 AND IOPL < 3 AND INT n)
            THEN
                #GP(0);
            ELSE (* protected mode or virtual-8086 mode interrupt *)
                GOTO PROTECTED-MODE;
        FI;
FI;

REAL-ADDRESS-MODE:
    IF ((DEST * 4) + 3) is not within IDT limit THEN #GP; FI;
    IF stack not large enough for a 6-byte return information THEN #SS; FI;
    Push (EFLAGS[15:0]);
    IF ← 0; (* Clear interrupt flag *)
    TF ← 0; (* Clear trap flag *)
    AC ← 0; (*Clear AC flag*)
    Push(CS);
    Push(IP);
    (* No error codes are pushed *)
    CS ← IDT(Descriptor (vector_number * 4), selector));
    EIP ← IDT(Descriptor (vector_number * 4), offset)); (* 16 bit offset AND 0000FFFFH *)
END;

PROTECTED-MODE:
    IF ((DEST * 8) + 7) is not within IDT limits
        OR selected IDT descriptor is not an interrupt-, trap-, or task-gate type
            THEN #GP((DEST * 8) + 2 + EXT);
            (* EXT is bit 0 in error code *)
    FI;
    IF software interrupt (* generated by INT n, INT 3, or INTO *)
        THEN
            IF gate descriptor DPL < CPL
                THEN #GP((vector_number * 8) + 2 );
                (* PE = 1, DPL<CPL, software interrupt *)
            FI;
    FI;
    IF gate not present THEN #NP((vector_number * 8) + 2 + EXT); FI;
    IF task gate (* specified in the selected interrupt table descriptor *)
        THEN GOTO TASK-GATE;
        ELSE GOTO TRAP-OR-INTERRUPT-GATE; (* PE = 1, trap/interrupt gate *)
    FI;
END;

TASK-GATE: (* PE = 1, task gate *)
    Read segment selector in task gate (IDT descriptor);
        IF local/global bit is set to local
            OR index not within GDT limits
```

```
                THEN #GP(TSS selector);
            FI;
            Access TSS descriptor in GDT;
            IF TSS descriptor specifies that the TSS is busy (low-order 5 bits set to 00001)
                THEN #GP(TSS selector);
            FI;
            IF TSS not present
                THEN #NP(TSS selector);
            FI;
        SWITCH-TASKS (with nesting) to TSS;
        IF interrupt caused by fault with error code
            THEN
                IF stack limit does not allow push of error code
                    THEN #SS(0);
                FI;
                Push(error code);
        FI;
        IF EIP not within code segment limit
            THEN #GP(0);
        FI;
END;
TRAP-OR-INTERRUPT-GATE
    Read segment selector for trap or interrupt gate (IDT descriptor);
    IF segment selector for code segment is null
        THEN #GP(0H + EXT); (* null selector with EXT flag set *)
    FI;
    IF segment selector is not within its descriptor table limits
        THEN #GP(selector + EXT);
    FI;
    Read trap or interrupt handler descriptor;
    IF descriptor does not indicate a code segment
        OR code segment descriptor DPL > CPL
            THEN #GP(selector + EXT);
    FI;
    IF trap or interrupt gate segment is not present,
        THEN #NP(selector + EXT);
    FI;
    IF code segment is non-conforming AND DPL < CPL
        THEN IF VM=0
            THEN
                GOTO INTER-PRIVILEGE-LEVEL-INTERRUPT;
                (* PE = 1, interrupt or trap gate, nonconforming *)
                (* code segment, DPL<CPL, VM = 0 *)
            ELSE (* VM = 1 *)
                IF code segment DPL ≠ 0 THEN #GP(new code segment selector); FI;
                GOTO INTERRUPT-FROM-VIRTUAL-8086-MODE;
                (* PE = 1, interrupt or trap gate, DPL<CPL, VM = 1 *)
        FI;
        ELSE (* PE = 1, interrupt or trap gate, DPL ≥ CPL *)
            IF VM = 1 THEN #GP(new code segment selector); FI;
            IF code segment is conforming OR code segment DPL = CPL
                THEN
                    GOTO INTRA-PRIVILEGE-LEVEL-INTERRUPT;
                ELSE
                    #GP(CodeSegmentSelector + EXT);
```

```
                    (* PE = 1, interrupt or trap gate, nonconforming *)
                    (* code segment, DPL>CPL *)
                FI;
    FI;
END;

INTER-PREVILEGE-LEVEL-INTERRUPT
    (* PE=1, interrupt or trap gate, non-conforming code segment, DPL<CPL *)
    (* Check segment selector and descriptor for stack of new privilege level in current TSS *)
    IF current TSS is 32-bit TSS
        THEN
            TSSstackAddress ← (new code segment DPL * 8) + 4
            IF (TSSstackAddress + 7) > TSS limit
                THEN #TS(current TSS selector); FI;
            NewSS ← TSSstackAddress + 4;
            NewESP ← stack address;
        ELSE (* TSS is 16-bit *)
            TSSstackAddress ← (new code segment DPL * 4) + 2
            IF (TSSstackAddress + 4) > TSS limit
                THEN #TS(current TSS selector); FI;
            NewESP ← TSSstackAddress;
            NewSS ← TSSstackAddress + 2;
    FI;
    IF segment selector is null THEN #TS(EXT); FI;
    IF segment selector index is not within its descriptor table limits
        OR segment selector's RPL ≠ DPL of code segment,
            THEN #TS(SS selector + EXT);
    FI;
Read segment descriptor for stack segment in GDT or LDT;
    IF stack segment DPL ≠ DPL of code segment,
        OR stack segment does not indicate writable data segment,
            THEN #TS(SS selector + EXT);
    FI;
    IF stack segment not present THEN #SS(SS selector+EXT); FI;
    IF 32-bit gate
        THEN
            IF new stack does not have room for 24 bytes (error code pushed)
                OR 20 bytes (no error code pushed)
                    THEN #SS(segment selector + EXT);
            FI;
        ELSE (* 16-bit gate *)
            IF new stack does not have room for 12 bytes (error code pushed)
                OR 10 bytes (no error code pushed);
                    THEN #SS(segment selector + EXT);
            FI;
    FI;
    IF instruction pointer is not within code segment limits THEN #GP(0); FI;
    SS:ESP ← TSS(NewSS:NewESP) (* segment descriptor information also loaded *)
    IF 32-bit gate
        THEN
            CS:EIP ← Gate(CS:EIP); (* segment descriptor information also loaded *)
        ELSE (* 16-bit gate *)
            CS:IP ← Gate(CS:IP); (* segment descriptor information also loaded *)
    FI;
    IF 32-bit gate
```

```
            THEN
                Push(far pointer to old stack); (* old SS and ESP, 3 words padded to 4 *);
                Push(EFLAGS);
                Push(far pointer to return instruction); (* old CS and EIP, 3 words padded to 4*);
                Push(ErrorCode); (* if needed, 4 bytes *)
            ELSE(* 16-bit gate *)
                Push(far pointer to old stack); (* old SS and SP, 2 words *);
                Push(EFLAGS(15..0]);
                Push(far pointer to return instruction); (* old CS and IP, 2 words *);
                Push(ErrorCode); (* if needed, 2 bytes *)
        FI;
        CPL ← CodeSegmentDescriptor(DPL);
        CS(RPL) ← CPL;
        IF interrupt gate
            THEN IF ← 0 (*interrupt flag set to 0: disabled*);
        FI;
        TF ← 0;
        VM ← 0;
        RF ← 0;
        NT ← 0;
    END;

    INTERRUPT-FROM-VIRTUAL-8086-MODE:
        (* Check segment selector and descriptor for privilege level 0 stack in current TSS *)
        IF current TSS is 32-bit TSS
            THEN
                TSSstackAddress ← (new code segment DPL * 8) + 4
                IF (TSSstackAddress + 7) > TSS limit
                    THEN #TS(current TSS selector); FI;
                NewSS ← TSSstackAddress + 4;
                NewESP ← stack address;
            ELSE (* TSS is 16-bit *)
                TSSstackAddress ← (new code segment DPL * 4) + 2
                IF (TSSstackAddress + 4) > TSS limit
                    THEN #TS(current TSS selector); FI;
                NewESP ← TSSstackAddress;
                NewSS ← TSSstackAddress + 2;
        FI;
            IF segment selector is null THEN #TS(EXT); FI;
            IF segment selector index is not within its descriptor table limits
                OR segment selector's RPL ≠ DPL of code segment,
                    THEN #TS(SS selector + EXT);
            FI;
        Access segment descriptor for stack segment in GDT or LDT;
        IF stack segment DPL ≠ DPL of code segment,
            OR stack segment does not indicate writable data segment,
                THEN #TS(SS selector + EXT);
        FI;
        IF stack segment not present
            THEN #SS(SS selector+EXT);
        FI;
        IF 32-bit gate
            THEN
                IF new stack does not have room for 40 bytes (error code pushed)
                    OR 36 bytes (no error code pushed);
```

```
                        THEN #SS(segment selector + EXT);
                FI;
            ELSE (* 16-bit gate *)
                IF new stack does not have room for 20 bytes (error code pushed)
                    OR 18 bytes (no error code pushed);
                        THEN #SS(segment selector + EXT);
                FI;
        FI;
        IF instruction pointer is not within code segment limits
            THEN #GP(0);
        FI;
        tempEFLAGS ← EFLAGS;
        VM ← 0;
        TF ← 0;
        RF ← 0;
        IF service through interrupt gate
            THEN IF = 0;
        FI;
        TempSS ← SS;
        TempESP ← ESP;
        SS:ESP ← TSS(SS0:ESP0); (* Change to level 0 stack segment *)
        (* Following pushes are 16 bits for 16-bit gate and 32 bits for 32-bit gates *)
        (* Segment selector pushes in 32-bit mode are padded to two words *)
        Push(GS);
        Push(FS);
        Push(DS);
        Push(ES);
        Push(TempSS);
        Push(TempESP);
        Push(TempEFlags);
        Push(CS);
        Push(EIP);
        GS ← 0; (*segment registers nullified, invalid in protected mode *)
        FS ← 0;
        DS ← 0;
        ES ← 0;
        CS ← Gate(CS);
        IF OperandSize = 32
            THEN
                EIP ← Gate(instruction pointer);
            ELSE (* OperandSize is 16 *)
                EIP ← Gate(instruction pointer) AND 0000FFFFH;
        FI;
        (* Starts execution of new routine in Protected Mode *)
    END;

    INTRA-PRIVILEGE-LEVEL-INTERRUPT:
        (* PE=1, DPL = CPL or conforming segment *)
        IF 32-bit gate
            THEN
                IF current stack does not have room for 16 bytes (error code pushed)
                    OR 12 bytes (no error code pushed); THEN #SS(0);
                FI;
            ELSE (* 16-bit gate *)
                IF current stack does not have room for 8 bytes (error code pushed)
```

```
                    OR 6 bytes (no error code pushed); THEN #SS(0);
                FI;
    FI;
    IF instruction pointer not within code segment limit
        THEN #GP(0);
    FI;
    IF 32-bit gate
        THEN
            Push (EFLAGS);
            Push (far pointer to return instruction); (* 3 words padded to 4 *)
            CS:EIP ← Gate(CS:EIP); (* segment descriptor information also loaded *)
            Push (ErrorCode); (* if any *)
        ELSE (* 16-bit gate *)
            Push (FLAGS);
            Push (far pointer to return location); (* 2 words *)
            CS:IP ← Gate(CS:IP); (* segment descriptor information also loaded *)
            Push (ErrorCode); (* if any *)
    FI;
    CS(RPL) ← CPL;
    IF interrupt gate
        THEN IF ← 0; (*interrupt flag set to 0: disabled*)
    FI;
    TF ← 0;
    NT ← 0;
    VM ← 0;
    RF ← 0;
END
```

## 17. Errors Corrected In Opcode Table

In the *Intel Architecture Software Developer's Manual*, Vol 2: *Instruction Set Reference*, Appendix A, Table A-2; multiple errors have been corrected. Rows with corrected cells are shown (note the bold text).

[table segments only, entire table not shown]

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| A | MOV | | | | **MOVS/ MOVSB Yb, Xb** | **MOVS/ MOVSW/ MOVSD Yv, Xv** | **CMPS/ CMPSB Yb, Xb** | **CMPS/ CMPSW/ CMPSD Xv, Yv** |
| | AL, Ob | eAX, Ov | Ob, AL | Ov, eAX | | | | |
| B | MOV immediate byte into byte register | | | | | | | |
| | AL | CL | DL | BL | AH | CH | DH | BH |
| C | Shift Grp 2[1A] | | **RET Iw** | **RET** | LES Gv, Mp | LDS Gv, Mp | Grp 11[1A] - MOV | |
| | Eb, Ib | Ev, Ib | | | | | Eb, Ib | Ev, Iv |