Addendum—Intel Architecture Software Developer’s Manual

Volume 2: Instruction Set Reference

Order Number: 243689-001

NOTE: The Intel Architecture Software Developer’s Manual consists of the following volumes: Basic Architecture, Order Number 243190; Addendum to the Basic Architecture (Order Number 243691); Instruction Set Reference, Order Number 243191; System Programming Guide, Order Number 243192; and the Addendum to the System Programming Guide, Order Number 243690.

Please refer to all of these volumes when evaluating your design needs.
Addendum—Intel Architecture Software Developer’s Manual

Volume 2: Instruction Set Reference

NOTE: The Intel Architecture Software Developer’s Manual consists of the following volumes: Basic Architecture, Order Number 243190; Addendum to the Basic Architecture (Order Number 243691); Instruction Set Reference, Order Number 243191; System Programming Guide, Order Number 243192; and the Addendum to the System Programming Guide, Order Number 243690.

Please refer to all of these volumes when evaluating your design needs.
# TABLE OF CONTENTS

## CHAPTER 3
### INSTRUCTION SET REFERENCE

3.1. INSTRUCTION REFERENCE ................................................................. 3-1
- FXRSTOR—Restore FP or MMX™ Technology State .......................... 3-2
- FXSAVE—Store FP or MMX™ Technology State ................................. 3-6
- SYSENTER—Fast Transition to System Call Entry Point ...................... 3-10
- SYSEXIT—Fast Transition from System Call Entry Point ................. 3-13
3.1. INSTRUCTION REFERENCE

This addendum provides detailed descriptions of four Intel Architecture instructions.
FXRSTOR—Restore FP or MMX™ Technology State

**Opcode** | **Instruction** | **Description**
---|---|---
0F AE, /1 | FXRSTOR m512byte | Load the FP or MMX™ technology state from m512byte

**Description**

The FXRSTOR instruction reloads the FP or MMX™ technology state (environment and registers) from the memory area defined by m512byte. This data should have been written by a previous FXSAVE.

The FP or MMX technology environment and registers consist of the following data structure (little-endian byte order as arranged in memory, with byte offset into row described by right column).
Three fields in the floating-point save area contain reserved bits that are not indicated in the

### FOP
The lower 11-bits contain the opcode, upper 5-bits are reserved.

### IP & DP
32-bit mode: 32-bit IP-offset.

16-bit mode: lower 16-bits are IP-offset and upper 16-bits are reserved.

The term, “Reserved,” is as defined in Section 1.4.2 of the *Intel Architecture Software Developer’s Manual, Volume 2: Instruction Set Reference*. Reserved bits are undefined, and using them risks incompatibility with future Intel Architecture processors. Furthermore, all “Reserved” fields in the tag word area should be set specifically to zero on a restore, or in cases where the software is attempting to initialize a floating-point context.

Unlike the FRSTOR instruction, FXRSTOR does not fault when loading an image from memory that contains a pending exception in the Floating-Point Status Word (FSW); only the next occurrence of this unmasked exception will result in the error condition being asserted. It also does not flush pending x87-FP exceptions. To check and raise exceptions when loading a new operating environment, use FWAIT after FXRSTOR.

### Operation

```c
FPUCtrlWord ← SRC(FPUCtrlWord);
FPUStratusWord ← SRC(FPUStratusWord);
FPUTagWord ← SRC(FPUTagWord);
FPUDataPointer ← SRC(FPUDataPointer);
FPUInstrPointer ← SRC(FPUInstrPointer);
FPUInstOpcode ← SRC(FPULastInstructionOpcode);
ST(0) ← SRC(ST(0));
ST(1) ← SRC(ST(1));
ST(2) ← SRC(ST(2));
ST(3) ← SRC(ST(3));
ST(4) ← SRC(ST(4));
ST(5) ← SRC(ST(5));
ST(6) ← SRC(ST(6));
ST(7) ← SRC(ST(7));
```
INSTRUCTION SET REFERENCE

Exceptions
#GP(0) If m512byte is not aligned on a 16-byte boundary.
#AC(0) If alignment check is enabled (CR0.AM = 1, EFLAGS.AC = 1 and CPL = 3) and m512byte is not aligned on a 16 byte boundary.
#UD If instruction is preceded by a lock prefix.

Numeric Exceptions
#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0) If a memory operand effective address is outside the SS segment limit.
#PF (fault-code) If a page fault occurs.
#NM If CR0.EM = 1 or CR0.TS = 1.
#AC If alignment check is enabled, and an unaligned memory reference is made while the current privilege level is 3.

Protected-Mode Exceptions
#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0) If a memory operand effective address is outside the SS segment limit.
#PF (fault-code) If a page fault occurs.
#NM If CR0.EM = 1 or CR0.TS = 1.
#AC If alignment check is enabled, and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions
Interrupt 13 If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM If CR0.EM = 1 or CR0.TS = 1.

Virtual-8086 Mode Exceptions
Same exceptions as in Real-Address Mode.
#PF (fault-code) If a page fault occurs.
#AC If alignment check is enabled, and an unaligned memory reference is made while the current privilege level is 3.
Notes

• State saved with FSAVE and restored with FXRSTOR (and vice versa) results in an incorrect restoration of state in the processor. Software should not depend on the behavior of the FXRSTOR instruction when it is preceded by either the REP, REPNE, or operand size override prefix. The application of these prefixes with FXRSTOR is defined as “reserved,” and processor behavior is model specific. Using these prefixes with FXRSTOR risks incompatibility with future Intel processors. The address size prefix has the usual effect on address calculation, but has no effect on the format of the FXRSTOR image.

• The FXRSTOR instruction assumes that the upper byte of the FPU Tag Word is equal to zero. If it is nonzero, the execution of the FXRSTOR instruction will cause an incorrect state to be generated in the processor.

Always ensure that FXRSTOR is used in conjunction with the FXSAVE instruction in a programming environment. Otherwise, ensure that the upper byte of the FPU Tag Word is zero before the FXRSTOR instruction is executed.

If an environment creates a condition where the upper byte of the FPU Tag Word is nonzero before execution of the FXRSTOR instruction, the result is an unpredictable system failure due to the loading of a corrupted state.
FXSAVE—Store FP or MMX™ Technology State

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F AE, /0</td>
<td>FXSAVE m512byte</td>
<td>Store FP or MMX™ technology state to m512byte</td>
</tr>
</tbody>
</table>

**Description**

The FXSAVE instruction writes the current FP or MMX technology state (environment and registers) to the specified destination defined by m512byte. It does this without checking for pending unmasked floating-point exceptions, similar to the operation of FNSAVE. Unlike the FSAVE/FNSAVE instructions, the processor retains the contents of the FP or MMX technology state in the processor after the particular state has been saved. This instruction has been optimized to maximize floating-point save performance.

The FXSAVE instruction is used when an operating system needs to perform a context switch or when an exception handler needs to use the FP and MMX technology units. It cannot be used by an application program to pass a “clean” FP state to a procedure, because it retains the current state. An application must explicitly execute an FINIT instruction after an FXSAVE to provide this functionality.

The save format is as described for the FXRSTOR instruction. All of the fields in bytes 0-160 retain the same internal format as the FSAVE instruction, except for the floating-point tag word (FTW). Unlike FSAVE, the FXSAVE instruction only saves the FTW valid bits rather than the entire x87-FP FTW field. The FTW bits are saved by FXSAVE in a non-TOS relative order, meaning that FR0 is always saved first, followed by FR1, FR2, and so forth.

As an example, if TOS=4 and only ST0, ST1 and ST2 are valid, FSAVE saves the FTW field in the following format:

```
ST3 ST2 ST1 ST0 ST7 ST6 ST5 ST4 (TOS=4)
FR7 FR6 FR5 FR4 FR3 FR2 FR1 FR0
11 xx xx xx 11 11 11 11
```

where xx is one of (00, 01, 10). A (11) indicates an Empty stack element. The values of 00, 01, and 10 indicate Valid, Zero, and Special, respectively. In this example, FXSAVE would save the following vector:

```
FR7 FR6 FR5 FR4 FR3 FR2 FR1 FR0
0 1 1 1 0 0 0 0
```
The FSAVE format for FTW can be recreated from the FTW valid bits and the stored 80-bit FP data (assuming the stored data was not the contents of MMX registers) using the following table.

<table>
<thead>
<tr>
<th>Exponent all 1s</th>
<th>Exponent all 0s</th>
<th>Fraction all 0s</th>
<th>J and M bits</th>
<th>FTW valid bit</th>
<th>x87 FTW</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0x 1x</td>
<td>1</td>
<td>Special 10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1x</td>
<td>1</td>
<td>Valid 00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00</td>
<td>1</td>
<td>Special 10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>Valid 00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0x</td>
<td>1</td>
<td>Special 10</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1x</td>
<td>1</td>
<td>Special 10</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>00</td>
<td>1</td>
<td>Zero 01</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>Special 10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1x</td>
<td>1</td>
<td>Special 10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1x</td>
<td>1</td>
<td>Special 10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>00</td>
<td>1</td>
<td>Special 10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>Special 10</td>
</tr>
</tbody>
</table>

For all legal combinations above 00 Empty 11

In binary floating-point format, a real number has three parts: a sign bit, a significand, and an exponent. The significand has two parts: a 1-bit binary integer (referred to as the J-bit) and a binary fraction.

The J-bit is defined to be the 1-bit binary integer to the left of the decimal place in the significand. The M-bit is defined to be the most significant bit of the fractional portion of the significand (i.e., the bit immediately to the right of the decimal place).

If the FXSAVE instruction is immediately preceded by an FP instruction which does not use a memory operand, then the FXSAVE instruction does not write/update the DP field, in the FXSAVE image.

The destination m512byte is assumed to be aligned on a 16-byte boundary. If m512byte is not aligned on a 16-byte boundary, FXSAVE generates a general protection exception.

**Operation**

(* Save FPU State and Registers *)

DEST(FPUC) ← FPUC;
DEST(FPUS) ← FPUS;
DEST(FPUT) ← Function of (FPUT);
DEST(FPUD) ← FPUD;
DEST(FPU) ← FPUD;
DEST(FPUL) ← FPUL;
DEST(ST(0)) ← ST(0);
DEST(ST(1)) ← ST(1);
DEST(ST(2)) ← ST(2);
INSTRUCTION SET REFERENCE

DEST(ST(3)) ← ST(3);
DEST(ST(4)) ← ST(4);
DEST(ST(5)) ← ST(5);
DEST(ST(6)) ← ST(6);
DEST(ST(7)) ← ST(7);
(* Does not initialize FPU -- Retains contents from above *)

Exceptions

#GP(0) If m512byte is not aligned on a 16-byte boundary.
#AC(0) If alignment check is enabled (CR0.AM = 1, EFLAGS.AC = 1 and CPL = 3) and m512byte is not aligned on a 16 byte boundary.
#UD If instruction is preceded by a lock prefix.

Numeric Exceptions

None.

Protected-Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0) If a memory operand effective address is outside the SS segment limit.
#PF (fault-code) If a page fault occurs.
#NM If CR0.EM = 1 or CR0.TS = 1.
#AC If alignment check is enabled, and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

Interrupt 13 If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM If CR0.EM = 1 or CR0.TS = 1.

Virtual-8086 Exceptions

Same exceptions as in Real-Address Mode
#PF (fault-code) If a page fault occurs.
#AC If alignment check is enabled, and an unaligned memory reference is made while the current privilege level is 3.
Notes

State saved with FXSAVE and restored with FRSTOR (and vice versa) results in an incorrect restoration of state in the processor. Software should not depend on the behavior of the FXRSTOR instruction when it is preceded by either the REP, REPNE, or operand size override prefix. The application of these prefixes with FXRSTOR is defined as “reserved,” and processor behavior is model specific. Using these prefixes with FXRSTOR risks incompatibility with future Intel processors. The address size prefix has the usual effect on address calculation, but has no effect on the format of the FXSAVE image.

If there is a pending unmasked FP exception at the time FXSAVE is executed, the sequence of FXSAVE-FWAIT-FXRSTOR results in an incorrect state in the processor. The FWAIT instruction causes the processor to check and handle pending unmasked FP exceptions. Since the processor does not clear the FPU state with FXSAVE, the exception is handled, but that fact is not reflected in the saved image. When the image is reloaded using FXRSTOR, the exception bits in the FSW get loaded incorrectly.
SYSENTER—Fast Transition to System Call Entry Point

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F, 34</td>
<td>SYSENTER</td>
<td>Transition to System Call Entry Point</td>
</tr>
</tbody>
</table>

Description

The SYSENTER instruction is part of the “Fast System Call” facility introduced on the Pentium® II processor. The SYSENTER instruction is optimized to provide the maximum performance for protection ring transitions to CPL = 0.

The SYSENTER instruction sets the following registers according to values specified by the operating system in certain model specific registers.

- CS register set to the value of (SYSENTER_CS_MSR)
- EIP register set to the value of (SYSENTER_EIP_MSR)
- SS register set to the sum of (8 plus the value in SYSENTER_CS_MSR)
- ESP register set to the value of (SYSENTER_ESP_MSR)

The processor does not save user stack or return address information, and does not save any registers.

The SYSENTER and SYSEXIT instructions do not constitute a call/return pair; therefore, the system call “stub” routines executed by user code (typically in shared libraries or DLLs) must perform the required register state save to create a system call/return pair.

The SYSENTER instruction always transfers to a flat protected-mode kernel at CPL = 0. SYSENTER can be invoked from all modes except real mode. The instruction requires that the following conditions are met by the operating system:

- The CS selector for the target ring 0 code segment is 32 bits, mapped as a flat 0-4 GB address space with execute and read permissions
- The SS selector for the target ring 0 stack segment is 32 bits, mapped as a flat 0-4 GB address space with read, write, and accessed permissions. This selector (Target Ring 0 SS Selector) is assigned the value of the new (CS selector + 8).

An operating system provides values for CS, EIP, SS, and ESP for the ring 0 entry point through use of model specific registers within the processor. These registers can be read from and written to by using the RDMSR and WRMSR instructions. The register addresses are defined to remain fixed at the following addresses on future processors that provide support for this feature.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSENTER_CS_MSR</td>
<td>Target Ring 0 CS Selector</td>
<td>174h</td>
</tr>
<tr>
<td>SYSENTER_ESP_MSR</td>
<td>Target Ring 0 ESP</td>
<td>175h</td>
</tr>
<tr>
<td>SYSENTER_EIP_MSR</td>
<td>Target Ring 0 Entry Point EIP</td>
<td>176h</td>
</tr>
</tbody>
</table>
The presence of this facility is indicated by the SYSENTER Present (SEP) bit 11 of CPUID. An operating system that detects the presence of the SEP bit must also qualify the processor family and model to ensure that the SYSENTER/SYSEXIT instructions are actually present. For example:

```c
If (CPUID SEP bit is set) {
    If (Family == 6) AND (Model < 3) AND (Stepping < 3) {
        THEN
            Fast System Call NOT supported
        }
    Else Fast System Call is supported
}
```

The Pentium Pro processor (Model = 1) returns a set SEP CPUID feature bit, but does not support the SYSENTER/SYSEXIT instructions.

**Operation**

**SYSENTER**

```c
IF CR0.PE == 0 THEN #GP(0)
IF SYSENTER_CS_MSR == 0 THEN #GP(0)

EFLAGS.VM := 0 // Prevent VM86 mode
EFLAGS.IF := 0 // Mask interrupts

CS.SEL := SYSENTER_CS_MSR // Operating system provides CS

// Set rest of CS to a fixed value
CS.SEL.CPL := 0 // CPL = 0
CS.SEL.BASE := 0 // Flat segment
CS.SEL.LIMIT := 0xFFFF // 4G limit
CS.SEL.G := 1 // 4 KB granularity
CS.SEL.S := 1
CS.SEL.TYPE_xCRA := 1011 // Execute + Read, Accessed
CS.SEL.D := 1 // 32 bit code
CS.SEL.DPL := 0
CS.SEL.RPL := 0
CS.SEL.P := 1
SS.SEL := CS.SEL+8

// Set rest of SS to a fixed value
SS.SEL.BASE := 0 // Flat segment
SS.SEL.LIMIT := 0xFFFF // 4G limit
SS.SEL.G := 1 // 4 KB granularity
SS.SEL.S := 1
SS.SEL.TYPE_xCRA := 0011 // Read/Write, Accessed
```
INSTRUCTION SET REFERENCE

SS.SEL.D := 1 // 32 bit stack
SS.SEL.DPL := 0
SS.SEL.RPL := 0
SS.SEL.P := 1

ESP := SYSENTER_ESP_MSR
EIP := SYSENTER_EIP_MSR

Exceptions
#GP(0) If SYSENTER_CS_MSR contains zero.

Numeric Exceptions
None.

Real-Address Mode Exceptions
#GP(0) If protected mode is not enabled.
SYSEXIT—Fast Transition from System Call Entry Point

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F, 35</td>
<td>SYSEXIT</td>
<td>Transition from System Call Entry Point</td>
</tr>
</tbody>
</table>

**Description**

The SYSEXIT instruction is part of the “Fast System Call” facility introduced on the Pentium II processor. The SYSEXIT instruction is optimized to provide the maximum performance for protection ring transitions from CPL = 0 to CPL = 3.

The SYSEXIT instruction sets the following registers according to values specified by the operating system in certain model specific or general purpose registers.

- **CS register**: set to the sum of (16 plus the value in SYSENTER_CS_MSR)
- **EIP register**: set to the value contained in the EDX register
- **SS register**: set to the sum of (24 plus the value in SYSENTER_CS_MSR)
- **ESP register**: set to the value contained in the ECX register

The processor does not save kernel stack or return address information, and does not save any registers.

The SYSENTER and SYSEXIT instructions do not constitute a call/return pair; therefore, the system call “stub” routines executed by user code (typically in shared libraries or DLLs) must perform the required register state restore to create a system call/return pair.

The SYSEXIT instruction always transfers to a flat protected-mode user at CPL = 3. SYSEXIT can be invoked only from protected mode and CPL = 0. The instruction requires that the following conditions are met by the operating system:

- The CS selector for the target ring 3 code segment is 32 bits, mapped as a flat 0-4 GB address space with execute, read, and nonconforming permissions.
- The SS selector for the target ring 3 stack segment is 32 bits, mapped as a flat 0-4 GB address space with expand-up, read, and write permissions.

An operating system must set the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS Selector</td>
<td>The Target Ring 3 CS Selector. This is assigned the sum of (16 + the value of SYSENTER_CS_MSR).</td>
</tr>
<tr>
<td>SS Selector</td>
<td>The Target Ring 3 SS Selector. This is assigned the sum of (24 + the value of SYSENTER_CS_MSR).</td>
</tr>
<tr>
<td>EIP</td>
<td>Target Ring 3 Return EIP. This is the target entry point, and is assigned the value contained in the EDX register.</td>
</tr>
<tr>
<td>ESP</td>
<td>Target Ring 3 Return ESP. This is the target entry point, and is assigned the value contained in the ECX register.</td>
</tr>
</tbody>
</table>
INSTRUCTION SET REFERENCE

The presence of this facility is indicated by the SYSENTER Present (SEP) bit 11 of CPUID. An operating system that detects the presence of the SEP bit must also qualify the processor family and model to ensure that the SYSENTER/SYSEXIT instructions are actually present, as described for the SYSENTER instruction. The Pentium Pro processor (Model = 1) returns a set SEP CPUID feature bit, but does not support the SYSENTER/SYSEXIT instructions.

Operation

SYSEXIT

IF SYSENTER_CS_MSR == 0 THEN #GP(0)
IF CR0.PE == 0 THEN #GP(0)
IF CPL <> 0 THEN #GP(0)

// Changing CS:EIP and SS:ESP is required

CS.SEL := (SYSENTER_CS_MSR + 16)  // Selector for return CS
CS.SEL.RPL := 3

// Set rest of CS to a fixed value
CS.SEL.BASE := 0  // Flat segment
CS.SEL.LIMIT := 0xFFFF  // 4G limit
CS.SEL.G := 1  // 4 KB granularity
CS.SEL.S := 1
CS.SEL.TYPE_xCRA := 1011  // Execute, Read, Nonconforming Code
CS.SEL.D := 1
CS.SEL.DPL := 3
CS.SEL.P := 1

SS.SEL := (SYSENTER_CS_MSR + 24)  // Selector for return SS
SS.SEL.RPL := 3

// Set rest of SS to a fixed value
SS.SEL.BASE := 0  // Flat segment
SS.SEL.LIMIT := 0xFFFF  // 4G limit
SS.SEL.G := 1  // 4 KB granularity
SS.SEL.S := 1
SS.SEL.TYPE_xCRA := 0011  // Expand Up, Read/Write, Data
SS.SEL.D := 1
SS.SEL.DPL := 3
SS.SEL.CPL := 3
SS.SEL.P := 1

ESP := ECX
EIP := EDX
Exceptions
#GP(0) If SYSENTER_CS_MSR contains zero.

Numeric Exceptions
None.

Protected-Mode Exceptions
#GP(0) If CPL is nonzero.

Real-Address Mode Exceptions
#GP(0) If protected mode is not enabled.