CHAPTER 5
INSTRUCTION SET SUMMARY

This chapter provides an abridged overview IA-32 instructions, divided into the following groups:

• General purpose
• x87 FPU
• x87 FPU and SIMD state management
• Intel MMX technology
• SSE extensions
• SSE2 extensions
• SSE3 extensions
• System instructions

Table 5-1 lists the groups and IA-32 processors that support each group. Within these groups, most instructions are collected into functional subgroups.

<table>
<thead>
<tr>
<th>Instruction Set Architecture</th>
<th>IA-32 Processor Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Purpose</td>
<td>All IA-32 processors</td>
</tr>
<tr>
<td>x87 FPU</td>
<td>Intel486, Pentium, Pentium with MMX Technology, Celeron, Pentium Pro, Pentium II, Pentium II Xeon, Pentium III, Pentium III Xeon, Pentium 4, Intel Xeon processors</td>
</tr>
<tr>
<td>x87 FPU and SIMD State Management</td>
<td>Pentium II, Pentium II Xeon, Pentium III, Pentium III Xeon, Pentium 4, Intel Xeon processors</td>
</tr>
<tr>
<td>MMX Technology</td>
<td>Pentium with MMX Technology, Celeron, Pentium II, Pentium II Xeon, Pentium III, Pentium III Xeon, Pentium 4, Intel Xeon processors</td>
</tr>
<tr>
<td>SSE Extensions</td>
<td>Pentium III, Pentium III Xeon, Pentium 4, Intel Xeon processors</td>
</tr>
<tr>
<td>SSE2 Extensions</td>
<td>Pentium 4, Intel Xeon processors</td>
</tr>
<tr>
<td>SSE3 Extensions</td>
<td>Pentium 4 supporting HT Technology (built on 90nm process technology)</td>
</tr>
<tr>
<td>System Instructions</td>
<td>All IA-32 processors</td>
</tr>
</tbody>
</table>

The following sections list instructions in each major group and subgroup. Given for each instruction is its mnemonic and descriptive names. When two or more mnemonics are given (for example, CMOVA/CMOVNBE), they represent different mnemonics for the same instruction opcode. Assemblers support redundant mnemonics for some instructions to make it easier to read.
code listings. For instance, CMOVA (Conditional move if above) and CMOVNBE (Conditional move if not below or equal) represent the same condition. For detailed information about a specific instruction, see Chapter 3, *Instruction Set Reference A-M* and Chapter 4, *Instruction Set Reference N-Z* of the *IA-32 Intel Architecture Software Developer’s Manual, Volumes 2A & 2B*.

5.1. **GENERAL-PURPOSE INSTRUCTIONS**

The general-purpose instructions preform basic data movement, arithmetic, logic, program flow, and string operations that programmers commonly use to write application and system software to run on IA-32 processors. They operate on data contained in memory, in the general-purpose registers (EAX, EBX, ECX, EDX, EDI, ESI, EBP, and ESP) and in the EFLAGS register. They also operate on address information contained in memory, the general-purpose registers, and the segment registers (CS, DS, SS, ES, FS, and GS).

This group of instructions includes the data transfer, binary integer arithmetic, decimal arithmetic, logic operations, shift and rotate, bit and byte operations, program control, string, flag control, segment register operations, and miscellaneous subgroups. The sections that following introduce each subgroup.

For more detailed information on general purpose-instructions, see Chapter 7, *Programming With the General-Purpose Instructions*.

5.1.1. **Data Transfer Instructions**

The data transfer instructions move data between memory and the general-purpose and segment registers. They also perform specific operations such as conditional moves, stack access, and data conversion.

- **MOV** Move data between general-purpose registers; move data between memory and general-purpose or segment registers; move immediates to general-purpose registers
- **CMOVE/CMOVZ** Conditional move if equal/Conditional move if zero
- **CMOVNE/CMOVNZ** Conditional move if not equal/Conditional move if not zero
- **CMOVA/CMOVNBE** Conditional move if above/Conditional move if not below or equal
- **CMOVAE/CMOVNB** Conditional move if above or equal/Conditional move if not below
- **CMOVBE/CMOVNAE** Conditional move if below/Conditional move if not above or equal
- **CMOVVG/CMOVNLE** Conditional move if greater/Conditional move if not less or equal
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMOVGE/CMOVNL</td>
<td>Conditional move if greater or equal/Conditional move if not less</td>
</tr>
<tr>
<td>CMOVVL/CMOVNGE</td>
<td>Conditional move if less/Conditional move if not greater or equal</td>
</tr>
<tr>
<td>CMOVLE/CMOVNG</td>
<td>Conditional move if less or equal/Conditional move if not greater</td>
</tr>
<tr>
<td>CMOVVC</td>
<td>Conditional move if carry</td>
</tr>
<tr>
<td>CMOVNC</td>
<td>Conditional move if not carry</td>
</tr>
<tr>
<td>CMOVO</td>
<td>Conditional move if overflow</td>
</tr>
<tr>
<td>CMOVNO</td>
<td>Conditional move if not overflow</td>
</tr>
<tr>
<td>CMOVS</td>
<td>Conditional move if sign (negative)</td>
</tr>
<tr>
<td>CMOVNS</td>
<td>Conditional move if not sign (non-negative)</td>
</tr>
<tr>
<td>CMOVVP/CMOVPE</td>
<td>Conditional move if parity/Conditional move if parity even</td>
</tr>
<tr>
<td>CMOVNP/CMOVPO</td>
<td>Conditional move if not parity/Conditional move if parity odd</td>
</tr>
<tr>
<td>XCHG</td>
<td>Exchange</td>
</tr>
<tr>
<td>BSWAP</td>
<td>Byte swap</td>
</tr>
<tr>
<td>XADD</td>
<td>Exchange and add</td>
</tr>
<tr>
<td>CMPXCHG</td>
<td>Compare and exchange</td>
</tr>
<tr>
<td>CMPXCHG8B</td>
<td>Compare and exchange 8 bytes</td>
</tr>
<tr>
<td>PUSH</td>
<td>Push onto stack</td>
</tr>
<tr>
<td>POP</td>
<td>Pop off of stack</td>
</tr>
<tr>
<td>PUSHA/PUSHAD</td>
<td>Push general-purpose registers onto stack</td>
</tr>
<tr>
<td>POPA/POPAD</td>
<td>Pop general-purpose registers from stack</td>
</tr>
<tr>
<td>CWD/CDQ</td>
<td>Convert word to doubleword/Convert doubleword to quadword</td>
</tr>
<tr>
<td>CBW/CWDE</td>
<td>Convert byte to word/Convert to doubleword in EAX register</td>
</tr>
<tr>
<td>MOV SX</td>
<td>Move and sign extend</td>
</tr>
<tr>
<td>MOV ZX</td>
<td>Move and zero extend</td>
</tr>
</tbody>
</table>
5.1.2. Binary Arithmetic Instructions

The binary arithmetic instructions perform basic binary integer computations on byte, word, and doubleword integers located in memory and/or the general purpose registers.

- **ADD** Integer add
- **ADC** Add with carry
- **SUB** Subtract
- **SBB** Subtract with borrow
- **IMUL** Signed multiply
- **MUL** Unsigned multiply
- **IDIV** Signed divide
- **DIV** Unsigned divide
- **INC** Increment
- **DEC** Decrement
- **NEG** Negate
- **CMP** Compare

5.1.3. Decimal Arithmetic Instructions

The decimal arithmetic instructions perform decimal arithmetic on binary coded decimal (BCD) data.

- **DAA** Decimal adjust after addition
- **DAS** Decimal adjust after subtraction
- **AAA** ASCII adjust after addition
- **AAS** ASCII adjust after subtraction
- **AAM** ASCII adjust after multiplication
- **AAD** ASCII adjust before division

5.1.4. Logical Instructions

The logical instructions perform basic AND, OR, XOR, and NOT logical operations on byte, word, and doubleword values.

- **AND** Perform bitwise logical AND
- **OR** Perform bitwise logical OR
INSTRUCTION SET SUMMARY

5.1.5. Shift and Rotate Instructions

The shift and rotate instructions shift and rotate the bits in word and doubleword operands.

- **SAR**  
  Shift arithmetic right
- **SHR**  
  Shift logical right
- **SAL/SHL**  
  Shift arithmetic left/Shift logical left
- **SHRD**  
  Shift right double
- **SHLD**  
  Shift left double
- **ROR**  
  Rotate right
- **ROL**  
  Rotate left
- **RCR**  
  Rotate through carry right
- **RCL**  
  Rotate through carry left

5.1.6. Bit and Byte Instructions

Bit instructions test and modify individual bits in word and doubleword operands. Byte instructions set the value of a byte operand to indicate the status of flags in the EFLAGS register.

- **BT**  
  Bit test
- **BTS**  
  Bit test and set
- **BTR**  
  Bit test and reset
- **BTC**  
  Bit test and complement
- **BSF**  
  Bit scan forward
- **BSR**  
  Bit scan reverse
- **SETE/SETZ**  
  Set byte if equal/Set byte if zero
- **SETNE/SETNZ**  
  Set byte if not equal/Set byte if not zero
- **SETA/SETNBE**  
  Set byte if above/Set byte if not below or equal
- **SETAE/SETNB/SETNC**  
  Set byte if above or equal/Set byte if not below/Set byte if not carry
- **SETB/SETNAE/SETC**  
  Set byte if below/Set byte if not above or equal/Set byte if carry
INSTRUCTION SET SUMMARY

- **SETBE/SETNA**: Set byte if below or equal/Set byte if not above
- **SETG/SETNLE**: Set byte if greater/Set byte if not less or equal
- **SETGE/SETNL**: Set byte if greater or equal/Set byte if not less
- **SETL/SETNGE**: Set byte if less/Set byte if not greater or equal
- **SETLE/SETNG**: Set byte if less or equal/Set byte if not greater
- **SETS**: Set byte if sign (negative)
- **SETNS**: Set byte if not sign (non-negative)
- **SETO**: Set byte if overflow
- **SETNO**: Set byte if not overflow
- **SETPE/SETP**: Set byte if parity even/Set byte if parity
- **SETPO/SETNP**: Set byte if parity odd/Set byte if not parity
- **TEST**: Logical compare

### 5.1.7. Control Transfer Instructions

The control transfer instructions provide jump, conditional jump, loop, and call and return operations to control program flow.

- **JMP**: Jump
- **JE/JZ**: Jump if equal/Jump if zero
- **JNE/JNZ**: Jump if not equal/Jump if not zero
- **JA/JNBE**: Jump if above/Jump if not below or equal
- **JAE/JNB**: Jump if above or equal/Jump if not below
- **JB/JNAE**: Jump if below/Jump if not above or equal
- **JBE/JNA**: Jump if below or equal/Jump if not above
- **JG/JNLE**: Jump if greater/Jump if not less or equal
- **JGE/JNL**: Jump if greater or equal/Jump if not less
- **JL/JNGE**: Jump if less/Jump if not greater or equal
- **JLE/JNG**: Jump if less or equal/Jump if not greater
- **JC**: Jump if carry
- **JNC**: Jump if not carry
- **JO**: Jump if overflow
- **JNO**: Jump if not overflow
INSTRUCTION SET SUMMARY

JS                  Jump if sign (negative)
JNS                 Jump if not sign (non-negative)
JPO/JNP             Jump if parity odd/Jump if not parity
JPE/JP              Jump if parity even/Jump if parity
JCXZ/JECXZ          Jump register CX zero/Jump register ECX zero
LOOP                Loop with ECX counter
LOOPZ/LOOPE         Loop with ECX and zero/Loop with ECX and equal
LOOPNZ/LOOPNE       Loop with ECX and not zero/Loop with ECX and not equal
CALL                Call procedure
RET                 Return
IRET                Return from interrupt
INT                 Software interrupt
INTO                Interrupt on overflow
BOUND               Detect value out of range
ENTER               High-level procedure entry
LEAVE               High-level procedure exit

5.1.8.       String Instructions

The string instructions operate on strings of bytes, allowing them to be moved to and from memory.

MOVS/MOVSB          Move string/Move byte string
MOVS/MOVSW          Move string/Move word string
MOVS/MOVSD          Move string/Move doubleword string
CMPS/CMPSB          Compare string/Compare byte string
CMPS/CMPSW          Compare string/Compare word string
CMPS/CMPSD          Compare string/Compare doubleword string
SCAS/SCASB          Scan string/Scan byte string
SCAS/SCASW          Scan string/Scan word string
SCAS/SCASD          Scan string/Scan doubleword string
LODS/LODSB          Load string/Load byte string
LODS/LODSW          Load string/Load word string
INSTRUCTION SET SUMMARY

LODS/LODSD  Load string/Load doubleword string
STOS/STOSB  Store string/Store byte string
STOS/STOSW  Store string/Store word string
STOS/STOSD  Store string/Store doubleword string
REP          Repeat while ECX not zero
REPE/REPZ    Repeat while equal/Repeat while zero
REPNE/REPNZ  Repeat while not equal/Repeat while not zero

5.1.9. I/O Instructions

These instructions move data between the processor’s I/O ports and a register or memory.
IN            Read from a port
OUT           Write to a port
INS/INSB     Input string from port/Input byte string from port
INS/INSW     Input string from port/Input word string from port
INS/INSD     Input string from port/Input doubleword string from port
OUTS/OUTSB   Output string to port/Output byte string to port
OUTS/OUTSW   Output string to port/Output word string to port
OUTS/OUTSD   Output string to port/Output doubleword string to port

5.1.10. Enter and Leave Instructions

These instructions provide machine-language support for procedure calls in block-structured languages.
ENTER        High-level procedure entry
LEAVE         High-level procedure exit

5.1.11. Flag Control (EFLAG) Instructions

The flag control instructions operate on the flags in the EFLAGS register.
STC           Set carry flag
CLC           Clear the carry flag
CMC           Complement the carry flag
CLD  Clear the direction flag
STD  Set direction flag
LAHF Load flags into AH register
SAHF Store AH register into flags
PUSHF/PUSHFD Push EFLAGS onto stack
POPF/POPFD Pop EFLAGS from stack
STI  Set interrupt flag
CLI  Clear the interrupt flag

5.1.12. Segment Register Instructions
The segment register instructions allow far pointers (segment addresses) to be loaded into the segment registers.

LDS  Load far pointer using DS
LES  Load far pointer using ES
LFS  Load far pointer using FS
LGS  Load far pointer using GS
LSS  Load far pointer using SS

5.1.13. Miscellaneous Instructions
The miscellaneous instructions provide such functions as loading an effective address, executing a "no-operation," and retrieving processor identification information.

LEA  Load effective address
NOP  No operation
UD2  Undefined instruction
XLAT/XLATB Table lookup translation
CPUID Processor Identification

5.2. X87 FPU INSTRUCTIONS
The x87 FPU instructions are executed by the processor’s x87 FPU. These instructions operate on floating-point, integer, and binary-coded decimal (BCD) operands. For more detail on x87 FPU instructions, see Chapter 8, Programming with the x87 FPU.
INSTRUCTION SET SUMMARY

These instructions are divided into the following subgroups: data transfer, load constants, and FPU control instructions. The sections that follow introduce each subgroup.

5.2.1. x87 FPU Data Transfer Instructions

The data transfer instructions move floating-point, integer, and BCD values between memory and the x87 FPU registers. They also perform conditional move operations on floating-point operands.

- **FLD**: Load floating-point value
- **FST**: Store floating-point value
- **FSTP**: Store floating-point value and pop
- **FILD**: Load integer
- **FIST**: Store integer
- **FISTP**: Store integer and pop
- **FBLD**: Load BCD
- **FBSTP**: Store BCD and pop
- **FXCH**: Exchange registers
- **FCMOVE**: Floating-point conditional move if equal
- **FCMOVNE**: Floating-point conditional move if not equal
- **FCMOVB**: Floating-point conditional move if below
- **FCMOVBE**: Floating-point conditional move if below or equal
- **FCMOVNB**: Floating-point conditional move if not below
- **FCMOVNBE**: Floating-point conditional move if not below or equal
- **FCMOVU**: Floating-point conditional move if unordered
- **FCMOVNU**: Floating-point conditional move if not unordered

5.2.2. x87 FPU Basic Arithmetic Instructions

The basic arithmetic instructions perform basic arithmetic operations on floating-point and integer operands.

- **FADD**: Add floating-point
- **FADDP**: Add floating-point and pop
- **FIADD**: Add integer

1. SSE3 provides an instruction FISTTP for integer conversion.
### INSTRUCTION SET SUMMARY

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSUB</td>
<td>Subtract floating-point</td>
</tr>
<tr>
<td>FSUBP</td>
<td>Subtract floating-point and pop</td>
</tr>
<tr>
<td>FISUB</td>
<td>Subtract integer</td>
</tr>
<tr>
<td>FSUBR</td>
<td>Subtract floating-point reverse</td>
</tr>
<tr>
<td>FSUBRP</td>
<td>Subtract floating-point reverse and pop</td>
</tr>
<tr>
<td>FISUBR</td>
<td>Subtract integer reverse</td>
</tr>
<tr>
<td>FMUL</td>
<td>Multiply floating-point</td>
</tr>
<tr>
<td>FMULP</td>
<td>Multiply floating-point and pop</td>
</tr>
<tr>
<td>FIMUL</td>
<td>Multiply integer</td>
</tr>
<tr>
<td>FDIV</td>
<td>Divide floating-point</td>
</tr>
<tr>
<td>FDIVP</td>
<td>Divide floating-point and pop</td>
</tr>
<tr>
<td>FIDIV</td>
<td>Divide integer</td>
</tr>
<tr>
<td>FDIVR</td>
<td>Divide floating-point reverse</td>
</tr>
<tr>
<td>FDIVRP</td>
<td>Divide floating-point reverse and pop</td>
</tr>
<tr>
<td>FIDIVR</td>
<td>Divide integer reverse</td>
</tr>
<tr>
<td>FPREM</td>
<td>Partial remainder</td>
</tr>
<tr>
<td>FPREM1</td>
<td>IEEE Partial remainder</td>
</tr>
<tr>
<td>FABS</td>
<td>Absolute value</td>
</tr>
<tr>
<td>FCHS</td>
<td>Change sign</td>
</tr>
<tr>
<td>FRNDINT</td>
<td>Round to integer</td>
</tr>
<tr>
<td>FSQUELE</td>
<td>Scale by power of two</td>
</tr>
<tr>
<td>FSQRT</td>
<td>Square root</td>
</tr>
<tr>
<td>FXTRACT</td>
<td>Extract exponent and significand</td>
</tr>
</tbody>
</table>

### 5.2.3. x87 FPU Comparison Instructions

The compare instructions examine or compare floating-point or integer operands.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCOM</td>
<td>Compare floating-point</td>
</tr>
<tr>
<td>FCOMP</td>
<td>Compare floating-point and pop</td>
</tr>
<tr>
<td>FCOMPP</td>
<td>Compare floating-point and pop twice</td>
</tr>
<tr>
<td>FUCOM</td>
<td>Unordered compare floating-point</td>
</tr>
<tr>
<td>FUCOMP</td>
<td>Unordered compare floating-point and pop</td>
</tr>
</tbody>
</table>
INSTRUCTION SET SUMMARY

FUCOMPP  Unordered compare floating-point and pop twice
FICOM    Compare integer
FICOMP   Compare integer and pop
FCOMI    Compare floating-point and set EFLAGS
FUCOMI   Unordered compare floating-point and set EFLAGS
FCOMIP   Compare floating-point, set EFLAGS, and pop
FUCOMIP  Unordered compare floating-point, set EFLAGS, and pop
FTST     Test floating-point (compare with 0.0)
FXAM     Examine floating-point

5.2.4. x87 FPU Transcendental Instructions
The transcendental instructions perform basic trigonometric and logarithmic operations on floating-point operands.

FSIN     Sine
FCOS     Cosine
FSINCOS  Sine and cosine
FPTAN    Partial tangent
FPATAN   Partial arctangent
F2XM1    \(2^x - 1\)
FYL2X    \(y \cdot \log_2 x\)
FYL2XP1  \(y \cdot \log_2 (x+1)\)

5.2.5. x87 FPU Load Constants Instructions
The load constants instructions load common constants, such as \(\pi\), into the x87 floating-point registers.

FLD1     Load +1.0
FLDZ     Load +0.0
FLDPI    Load \(\pi\)
FLDL2E   Load \(\log_2 e\)
FLDLN2   Load \(\log_2 2\)
FLDL2T   Load \(\log_2 10\)
FLDLG2   Load \(\log_{10} 2\)
5.2.6. **x87 FPU Control Instructions**

The x87 FPU control instructions operate on the x87 FPU register stack and save and restore the x87 FPU state.

- **FINCSTP** Increment FPU register stack pointer
- **FDECSTP** Decrement FPU register stack pointer
- **FFREE** Free floating-point register
- **FINIT** Initialize FPU after checking error conditions
- **FNINIT** Initialize FPU without checking error conditions
- **FCLEX** Clear floating-point exception flags after checking for error conditions
- **FNCLEX** Clear floating-point exception flags without checking for error conditions
- **FSTCW** Store FPU control word after checking error conditions
- **FNSTCW** Store FPU control word without checking error conditions
- **FLDCW** Load FPU control word
- **FSTENV** Store FPU environment after checking error conditions
- **FNSTENV** Store FPU environment without checking error conditions
- **FLDENV** Load FPU environment
- **FSAVE** Save FPU state after checking error conditions
- **FNSAVE** Save FPU state without checking error conditions
- **FRSTOR** Restore FPU state
- **FSTSW** Store FPU status word after checking error conditions
- **FNSTSW** Store FPU status word without checking error conditions
- **WAIT/FWAIT** Wait for FPU
- **FNOP** FPU no operation
5.3. X87 FPU AND SIMD STATE MANAGEMENT INSTRUCTIONS

Two state management instructions were introduced into the IA-32 architecture with the Pentium II processor family:

- FXSAVE: Save x87 FPU and SIMD state
- FXRSTOR: Restore x87 FPU and SIMD state

Initially, these instructions operated only on the x87 FPU (and MMX) registers to perform a fast save and restore, respectively, of the x87 FPU and MMX state. With the introduction of SSE extensions in the Pentium III processor family, these instructions were expanded to also save and restore the state of the XMM and MXCSR registers.

See Section 10.5., “FXSAVE and FXRSTOR Instructions” for more detail.

5.4. MMX™ INSTRUCTIONS

Four extensions have been introduced into the IA-32 architecture to permit IA-32 processors to perform single-instruction multiple-data (SIMD) operations. These extensions include the MMX technology, SSE extensions, SSE2 extensions, and SSE3 extensions. For a discussion that puts SIMD instructions in their historical context, see Section 2.3., “SIMD Instructions”.

MMX instructions operate on packed byte, word, doubleword, or quadword integer operands contained in memory, in MMX registers, and/or in general-purpose registers. For more detail on these instructions, see Chapter 9, Programming with Intel® MMX™ Technology.

MMX instructions can only be executed on IA-32 processors that support the MMX technology. Support for these instructions can be detected with the CPUID instruction. See the description of the CPUID instruction in Chapter 3, Instruction Set Reference A-M of the IA-32 Intel Architecture Software Developer’s Manual, Volume 2A.

MMX instructions are divided into the following subgroups: data transfer, conversion, packed arithmetic, comparison, logical, shift and rotate, and state management instructions. The sections that follow introduce each subgroup.

5.4.1. MMX Data Transfer Instructions

The data transfer instructions move doubleword and quadword operands between MMX registers and between MMX registers and memory.

- MOVD: Move doubleword
- MOVQ: Move quadword
5.4.2. MMX Conversion Instructions

The conversion instructions pack and unpack bytes, words, and doublewords.

- **PACKSSWB**: Pack words into bytes with signed saturation.
- **PACKSSDW**: Pack doublewords into words with signed saturation.
- **PACKUSWB**: Pack words into bytes with unsigned saturation.
- **PUNPCKHBW**: Unpack high-order bytes.
- **PUNPCKHWD**: Unpack high-order words.
- **PUNPCKHDQ**: Unpack high-order doublewords.
- **PUNPCKLBW**: Unpack low-order bytes.
- **PUNPCKLWD**: Unpack low-order words.
- **PUNPCKLDQ**: Unpack low-order doublewords.

5.4.3. MMX Packed Arithmetic Instructions

The packed arithmetic instructions perform packed integer arithmetic on packed byte, word, and doubleword integers.

- **PADD**: Add packed byte integers.
- **PADDW**: Add packed word integers.
- **PADD**: Add packed doubleword integers.
- **PADDKB**: Add packed signed byte integers with signed saturation.
- **PADDKW**: Add packed signed word integers with signed saturation.
- **PADDUSB**: Add packed unsigned byte integers with unsigned saturation.
- **PADDUSW**: Add packed unsigned word integers with unsigned saturation.
- **PSUB**: Subtract packed byte integers.
- **PSUBW**: Subtract packed word integers.
- **PSUBD**: Subtract packed doubleword integers.
- **PSUBSB**: Subtract packed signed byte integers with signed saturation.
- **PSUBSW**: Subtract packed signed word integers with signed saturation.
- **PSUBUSB**: Subtract packed unsigned byte integers with unsigned saturation.
- **PSUBUSW**: Subtract packed unsigned word integers with unsigned saturation.
- **PMULHW**: Multiply packed signed word integers and store high result.
INSTRUCTION SET SUMMARY

PMULLW Multiply packed signed word integers and store low result
PMADDWD Multiply and add packed word integers

5.4.4. MMX Comparison Instructions
The compare instructions compare packed bytes, words, or doublewords.

PCMPEQB Compare packed bytes for equal
PCMPEQW Compare packed words for equal
PCMPEQD Compare packed doublewords for equal
PCMPGTB Compare packed signed byte integers for greater than
PCMPGTW Compare packed signed word integers for greater than
PCMPGTD Compare packed signed doubleword integers for greater than

5.4.5. MMX Logical Instructions
The logical instructions perform AND, AND NOT, OR, and XOR operations on quadword operands.
PAND Bitwise logical AND
PANDN Bitwise logical AND NOT
POR Bitwise logical OR
PXOR Bitwise logical exclusive OR

5.4.6. MMX Shift and Rotate Instructions
The shift and rotate instructions shift and rotate packed bytes, words, or doublewords, or quadwords in 64-bit operands.

PSLLW Shift packed words left logical
PSLLD Shift packed doublewords left logical
PSLLQ Shift packed quadword left logical
PSRLW Shift packed words right logical
PSRLD Shift packed doublewords right logical
PSRLQ Shift packed quadword right logical
PSRAW Shift packed words right arithmetic
PSRAD Shift packed doublewords right arithmetic
5.4.7. **MMX State Management Instructions**

The EMMS instruction clears the MMX state from the MMX registers.

EMMS                      Empty MMX state

5.5. **SSE INSTRUCTIONS**

SSE instructions represent an extension of the SIMD execution model introduced with the MMX technology. For more detail on these instructions, see Chapter 10, *Programming with Streaming SIMD Extensions (SSE)*.

SSE instructions can only be executed on IA-32 processors that support SSE extensions. Support for these instructions can be detected with the CPUID instruction (see the description of the CPUID instruction in Chapter 3, *Instruction Set Reference A-M of the IA-32 Intel Architecture Software Developer’s Manual, Volume 2A*).

SSE instructions are divided into four subgroups (note that the first subgroup has subordinate subgroups of its own):

- SIMD single-precision floating-point instructions that operate on the XMM registers
- MXSCR state management instructions
- 64-bit SIMD integer instructions that operate on the MMX registers
- Cacheability control, prefetch, and instruction ordering instructions

The following sections provide an overview of these groups.

5.5.1. **SSE SIMD Single-Precision Floating-Point Instructions**

These instructions operate on packed and scalar single-precision floating-point values located in XMM registers and/or memory. This subgroup is further divided into the following subordinate subgroups: data transfer, packed arithmetic, comparison, logical, shuffle and unpack, and conversion instructions.

5.5.1.1. **SSE Data Transfer Instructions**

SSE data transfer instructions move packed and scalar single-precision floating-point operands between XMM registers and between XMM registers and memory.

- **MOVAPS**                      Move four aligned packed single-precision floating-point values between XMM registers or between and XMM register and memory
- **MOVUPS**                      Move four unaligned packed single-precision floating-point values between XMM registers or between and XMM register and memory
- **MOVHPS**                      Move two packed single-precision floating-point values to an from the high quadword of an XMM register and memory
### INSTRUCTION SET SUMMARY

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVHLPS</td>
<td>Move two packed single-precision floating-point values from the high quadword of an XMM register to the low quadword of another XMM register</td>
</tr>
<tr>
<td>MOVLPS</td>
<td>Move two packed single-precision floating-point values to an from the low quadword of an XMM register and memory</td>
</tr>
<tr>
<td>MOVLHPS</td>
<td>Move two packed single-precision floating-point values from the low quadword of an XMM register to the high quadword of another XMM register</td>
</tr>
<tr>
<td>MOVMSKPS</td>
<td>Extract sign mask from four packed single-precision floating-point values</td>
</tr>
<tr>
<td>MOVSS</td>
<td>Move scalar single-precision floating-point value between XMM registers or between an XMM register and memory</td>
</tr>
</tbody>
</table>

#### 5.5.1.2. SSE Packed Arithmetic Instructions

SSE packed arithmetic instructions perform packed and scalar arithmetic operations on packed and scalar single-precision floating-point operands.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDPS</td>
<td>Add packed single-precision floating-point values</td>
</tr>
<tr>
<td>ADDSS</td>
<td>Add scalar single-precision floating-point values</td>
</tr>
<tr>
<td>SUBPS</td>
<td>Subtract packed single-precision floating-point values</td>
</tr>
<tr>
<td>SUBSS</td>
<td>Subtract scalar single-precision floating-point values</td>
</tr>
<tr>
<td>MULPS</td>
<td>Multiply packed single-precision floating-point values</td>
</tr>
<tr>
<td>MULSS</td>
<td>Multiply scalar single-precision floating-point values</td>
</tr>
<tr>
<td>DIVPS</td>
<td>Divide packed single-precision floating-point values</td>
</tr>
<tr>
<td>DIVSS</td>
<td>Divide scalar single-precision floating-point values</td>
</tr>
<tr>
<td>RCPAPS</td>
<td>Compute reciprocals of packed single-precision floating-point values</td>
</tr>
<tr>
<td>RCPSS</td>
<td>Compute reciprocal of scalar single-precision floating-point values</td>
</tr>
<tr>
<td>SQRTPS</td>
<td>Compute square roots of packed single-precision floating-point values</td>
</tr>
<tr>
<td>SQRTSS</td>
<td>Compute square root of scalar single-precision floating-point values</td>
</tr>
<tr>
<td>RSRTPS</td>
<td>Compute reciprocals of square roots of packed single-precision floating-point values</td>
</tr>
<tr>
<td>RSQRTSS</td>
<td>Compute reciprocal of square root of scalar single-precision floating-point values</td>
</tr>
<tr>
<td>MAXPS</td>
<td>Return maximum packed single-precision floating-point values</td>
</tr>
</tbody>
</table>
### INSTRUCTION SET SUMMARY

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXSS</td>
<td>Return maximum scalar single-precision floating-point values</td>
</tr>
<tr>
<td>MINPS</td>
<td>Return minimum packed single-precision floating-point values</td>
</tr>
<tr>
<td>MINSS</td>
<td>Return minimum scalar single-precision floating-point values</td>
</tr>
</tbody>
</table>

#### 5.5.1.3. SSE Comparison Instructions

SSE compare instructions compare packed and scalar single-precision floating-point operands.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPPS</td>
<td>Compare packed single-precision floating-point values</td>
</tr>
<tr>
<td>CMPSS</td>
<td>Compare scalar single-precision floating-point values</td>
</tr>
<tr>
<td>COMISS</td>
<td>Perform ordered comparison of scalar single-precision floating-point values and set flags in EFLAGS register</td>
</tr>
<tr>
<td>UCOMISS</td>
<td>Perform unordered comparison of scalar single-precision floating-point values and set flags in EFLAGS register</td>
</tr>
</tbody>
</table>

#### 5.5.1.4. SSE Logical Instructions

SSE logical instructions perform bitwise AND, AND NOT, OR, and XOR operations on packed single-precision floating-point operands.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANDPS</td>
<td>Perform bitwise logical AND of packed single-precision floating-point values</td>
</tr>
<tr>
<td>ANDNPS</td>
<td>Perform bitwise logical AND NOT of packed single-precision floating-point values</td>
</tr>
<tr>
<td>ORPS</td>
<td>Perform bitwise logical OR of packed single-precision floating-point values</td>
</tr>
<tr>
<td>XORPS</td>
<td>Perform bitwise logical XOR of packed single-precision floating-point values</td>
</tr>
</tbody>
</table>

#### 5.5.1.5. SSE Shuffle and Unpack Instructions

SSE shuffle and unpack instructions shuffle or interleave single-precision floating-point values in packed single-precision floating-point operands.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHUFPS</td>
<td>Shuffles values in packed single-precision floating-point operands</td>
</tr>
<tr>
<td>UNPCKHPS</td>
<td>Unpacks and interleaves the two high-order values from two single-precision floating-point operands</td>
</tr>
<tr>
<td>UNPCKLPS</td>
<td>Unpacks and interleaves the two low-order values from two single-precision floating-point operands</td>
</tr>
</tbody>
</table>
5.5.1.6. **SSE Conversion Instructions**

SSE conversion instructions convert packed and individual doubleword integers into packed and scalar single-precision floating-point values and vice versa.

- `CVTPI2PS` Convert packed doubleword integers to packed single-precision floating-point values
- `CVTSI2SS` Convert doubleword integer to scalar single-precision floating-point value
- `CVTPS2PI` Convert packed single-precision floating-point values to packed doubleword integers
- `CVTTPS2PI` Convert with truncation packed single-precision floating-point values to packed doubleword integers
- `CVTSS2SI` Convert a scalar single-precision floating-point value to a doubleword integer
- `CVTTSS2SI` Convert with truncation a scalar single-precision floating-point value to a scalar doubleword integer

5.5.2. **SSE MXCSR State Management Instructions**

MXCSR state management instructions allow saving and restoring the state of the MXCSR control and status register.

- `LDMXCSR` Load MXCSR register
- `STMXCSR` Save MXCSR register state

5.5.3. **SSE 64-Bit SIMD Integer Instructions**

These SSE 64-bit SIMD integer instructions perform additional operations on packed bytes, words, or doublewords contained in MMX registers. They represent enhancements to the MMX instruction set described in Section 5.4., “MMX™ Instructions”.

- `PAVGB` Compute average of packed unsigned byte integers
- `PAVGW` Compute average of packed unsigned byte integers
- `PEXTRW` Extract word
- `PINSRW` Insert word
- `PMAXUB` Maximum of packed unsigned byte integers
- `PMAWSW` Maximum of packed signed word integers
- `PMINUB` Minimum of packed unsigned byte integers
- `PMINSW` Minimum of packed signed word integers
INSTRUCTION SET SUMMARY

PMOVMSKB  Move byte mask
PMULHUW  Multiply packed unsigned integers and store high result
PSADBW  Compute sum of absolute differences
PSHUFW  Shuffle packed integer word in MMX register

5.5.4.  SSE Cacheability Control, Prefetch, and Instruction Ordering Instructions

The cacheability control instructions provide control over the caching of non-temporal data when storing data from the MMX and XMM registers to memory. The PREFETCHh allows data to be prefetched to a selected cache level. The SFENCE instruction controls instruction ordering on store operations.

MASKMOVQ  Non-temporal store of selected bytes from an MMX register into memory
MOVNTQ  Non-temporal store of quadword from an MMX register into memory
MOVNTPS  Non-temporal store of four packed single-precision floating-point values from an XMM register into memory
PREFETCHh  Load 32 or more of bytes from memory to a selected level of the processor’s cache hierarchy
SFENCE  Serializes store operations

5.6.  SSE2 INSTRUCTIONS

SSE2 extensions represent an extension of the SIMD execution model introduced with MMX technology and the SSE extensions. SSE2 instructions operate on packed double-precision floating-point operands and on packed byte, word, doubleword, and quadword operands located in the XMM registers. For more detail on these instructions, see Chapter 11, Programming with Streaming SIMD Extensions 2 (SSE2).

SSE2 instructions can only be executed on IA-32 processors that support the SSE2 extensions. Support for these instructions can be detected with the CPUID instruction (see the description of the CPUID instruction in Chapter 3, Instruction Set Reference A-M of the IA-32 Intel Architecture Software Developer’s Manual, Volume 2A).

These instructions are divided into four subgroups (note that the first subgroup is further divided into subordinate subgroups):

• Packed and scalar double-precision floating-point instructions
• Packed single-precision floating-point conversion instructions
• 128-bit SIMD integer instructions
INSTRUCTION SET SUMMARY

- Cacheability-control and instruction ordering instructions

The following sections give an overview of each subgroup.

5.6.1. **SSE2 Packed and Scalar Double-Precision Floating-Point Instructions**

SSE2 packed and scalar double-precision floating-point instructions are divided into the following subordinate subgroups: data movement, arithmetic, comparison, conversion, logical, and shuffle operations on double-precision floating-point operands. These are introduced in the sections that follow.

5.6.1.1. **SSE2 Data Movement Instructions**

SSE2 data movement instructions move double-precision floating-point data between XMM registers and between XMM registers and memory.

- **MOVAPD** Move two aligned packed double-precision floating-point values between XMM registers or between an XMM register and memory
- **MOVUPD** Move two unaligned packed double-precision floating-point values between XMM registers or between an XMM register and memory
- **MOVHPD** Move high packed double-precision floating-point value to an from the high quadword of an XMM register and memory
- **MOVLPD** Move low packed single-precision floating-point value to an from the low quadword of an XMM register and memory
- **MOVMSKPD** Extract sign mask from two packed double-precision floating-point values
- **MOVSD** Move scalar double-precision floating-point value between XMM registers or between an XMM register and memory

5.6.1.2. **SSE2 Packed Arithmetic Instructions**

The arithmetic instructions perform addition, subtraction, multiply, divide, square root, and maximum/minimum operations on packed and scalar double-precision floating-point operands.

- **ADDPD** Add packed double-precision floating-point values
- **ADDSD** Add scalar double precision floating-point values
- **SUBPD** Subtract scalar double-precision floating-point values
- **SUBSD** Subtract scalar double-precision floating-point values
- **MULPD** Multiply packed double-precision floating-point values
- **MULSD** Multiply scalar double-precision floating-point values
DIVPD       Divide packed double-precision floating-point values
DIVSD       Divide scalar double-precision floating-point values
SQRTPD      Compute packed square roots of packed double-precision floating-point values
SQRRTSD     Compute scalar square root of scalar double-precision floating-point values
MAXPD       Return maximum packed double-precision floating-point values
MAXSD       Return maximum scalar double-precision floating-point values
MINPD       Return minimum packed double-precision floating-point values
MINSD       Return minimum scalar double-precision floating-point values

5.6.1.3.   SSE2 Logical Instructions

SSE2 logical instructions preform AND, AND NOT, OR, and XOR operations on packed double-precision floating-point values.

ANDPD       Perform bitwise logical AND of packed double-precision floating-point values
ANDNPD      Perform bitwise logical AND NOT of packed double-precision floating-point values
ORPD        Perform bitwise logical OR of packed double-precision floating-point values
XORPD       Perform bitwise logical XOR of packed double-precision floating-point values

5.6.1.4.   SSE2 Compare Instructions

SSE2 compare instructions compare packed and scalar double-precision floating-point values and return the results of the comparison either to the destination operand or to the EFLAGS register.

CMPPPD      Compare packed double-precision floating-point values
CMPPSD      Compare scalar double-precision floating-point values
COMISD      Perform ordered comparison of scalar double-precision floating-point values and set flags in EFLAGS register
UCOMISD     Perform unordered comparison of scalar double-precision floating-point values and set flags in EFLAGS register.
5.6.1.5. **SSE2 Shuffle and Unpack Instructions**

SSE2 shuffle and unpack instructions shuffle or interleave double-precision floating-point values in packed double-precision floating-point operands.

- **SHUFPD**: Shuffles values in packed double-precision floating-point operands.
- **UNPCKHPD**: Unpacks and interleaves the high values from two packed double-precision floating-point operands.
- **UNPCKLPD**: Unpacks and interleaves the low values from two packed double-precision floating-point operands.

5.6.1.6. **SSE2 Conversion Instructions**

SSE2 conversion instructions convert packed and individual doubleword integers into packed and scalar double-precision floating-point values and vice versa. They also convert between packed and scalar single-precision and double-precision floating-point values.

- **CVTPD2PI**: Convert packed double-precision floating-point values to packed doubleword integers.
- **CVTTPD2PI**: Convert with truncation packed double-precision floating-point values to packed doubleword integers.
- **CVTPD2Q**: Convert packed double-precision floating-point values to packed doubleword integers.
- **CVTDQ2PD**: Convert packed doubleword integers to packed double-precision floating-point values.
- **CVTPD2PS**: Convert packed double-precision floating-point values to packed single-precision floating-point values.
- **CVTSS2SD**: Convert scalar single-precision floating-point values to scalar double-precision floating-point values.
- **CVTSD2SS**: Convert scalar double-precision floating-point values to scalar single-precision floating-point values.
- **CVTSD2SI**: Convert scalar double-precision floating-point values to a double-word integer.
- **CVTTSD2SI**: Convert with truncation scalar double-precision floating-point values to scalar doubleword integers.
CVTSI2SD Convert doubleword integer to scalar double-precision floating-point value

5.6.2. SSE2 Packed Single-Precision Floating-Point Instructions

SSE2 packed single-precision floating-point instructions perform conversion operations on single-precision floating-point and integer operands. These instructions represent enhancements to the SSE single-precision floating-point instructions.

CVTDQ2PS Convert packed doubleword integers to packed single-precision floating-point values
CVTPS2DQ Convert packed single-precision floating-point values to packed doubleword integers
CVTTPS2DQ Convert with truncation packed single-precision floating-point values to packed doubleword integers

5.6.3. SSE2 128-Bit SIMD Integer Instructions

SSE2 SIMD integer instructions perform additional operations on packed words, doublewords, and quadwords contained in XMM and MMX registers

MOVDQA Move aligned double quadword.
MOVDQU Move unaligned double quadword
MOVQ2DQ Move quadword integer from MMX to XMM registers
MOVDQ2Q Move quadword integer from XMM to MMX registers
PMULUDQ Multiply packed unsigned doubleword integers
PADDQ Add packed quadword integers
PSUBQ Subtract packed quadword integers
PSHUFLW Shuffle packed low words
PSHUFHW Shuffle packed high words
PSHUFD Shuffle packed doublewords
PSLLDQ Shift double quadword left logical
PSRLDQ Shift double quadword right logical
PUNPCKHQLDQ Unpack high quadwords
PUNPCKLQDQ Unpack low quadwords
5.6.4. **SSE2 Cacheability Control and Ordering Instructions**

SSE2 cacheability control instructions provide additional operations for caching of non-temporal data when storing data from XMM registers to memory. LFENCE and MFENCE provide additional control of instruction ordering on store operations.

- **CLFLUSH** Flushes and invalidates a memory operand and its associated cache line from all levels of the processor’s cache hierarchy
- **LFENCE** Serializes load operations
- **MFENCE** Serializes load and store operations
- **PAUSE** Improves the performance of “spin-wait loops”
- **MASKMOVDQU** Non-temporal store of selected bytes from an XMM register into memory
- **MOVNTPD** Non-temporal store of two packed double-precision floating-point values from an XMM register into memory
- **MOVNTDQ** Non-temporal store of double quadword from an XMM register into memory
- **MOVNTI** Non-temporal store of a doubleword from a general-purpose register into memory

5.7. **SSE3 INSTRUCTIONS**

The SSE3 extensions offers 13 instructions that accelerate performance of Streaming SIMD Extensions technology, Streaming SIMD Extensions 2 technology, and x87-FP math capabilities. These instructions can be grouped into the following categories:

- One x87FPU instruction used in integer conversion
- One SIMD integer instruction that addresses unaligned data loads
- Two SIMD floating-point packed ADD/SUB instructions
- Four SIMD floating-point horizontal ADD/SUB instructions
- Three SIMD floating-point LOAD/MOVE/DUPLICATE instructions
- Two thread synchronization instructions

SSE3 instructions can only be executed on IA-32 processors that support SSE3 extensions. Support for these instructions can be detected with the CPUID instruction (see the description of the CPUID instruction in Chapter 3, *Instruction Set Reference A-M* of the *IA-32 Intel Architecture Software Developer’s Manual, Volume 2A*).

The sections that follow describe each subgroup.
### 5.7.1. SSE x87-FP Integer Conversion Instruction

**FISTTP**
Behaves like the FISTP instruction but uses truncation, irrespective of the rounding mode specified in the floating-point control word (FCW)

### 5.7.2. SSE3 Specialized 128-bit Unaligned Data Load Instruction

**LDDQU**
Special 128-bit unaligned load designed to avoid cache line splits

### 5.7.3. SSE3 SIMD Floating-Point Packed ADD/SUB Instructions

**ADDSUBPS**
Performs single-precision addition on the second and fourth pairs of 32-bit data elements within the operands; single-precision subtraction on the first and third pairs

**ADDSUBPD**
Performs double-precision addition on the second pair of quadwords, and double-precision subtraction on the first pair

### 5.7.4. SSE3 SIMD Floating-Point Horizontal ADD/SUB Instructions

**HADDPS**
Performs a single-precision addition on contiguous data elements. The first data element of the result is obtained by adding the first and second elements of the first operand; the second element by adding the third and fourth elements of the first operand; the third by adding the first and second elements of the second operand; and the fourth by adding the third and fourth elements of the second operand.

**HSUBPS**
Performs a single-precision subtraction on contiguous data elements. The first data element of the result is obtained by subtracting the second element of the first operand from the first element of the first operand; the second element by subtracting the fourth element of the first operand from the third element of the first operand; the third by subtracting the second element of the second operand from the first element of the second operand; and the fourth by subtracting the fourth element of the second operand from the third element of the second operand.

**HADDPD**
Performs a double-precision addition on contiguous data elements. The first data element of the result is obtained by adding the first and second elements of the first operand; the second element by adding the first and second elements of the second operand.

**HSUBPD**
Performs a double-precision subtraction on contiguous data elements. The first data element of the result is obtained by...
subtracting the second element of the first operand from the first element of the first operand; the second element by subtracting the second element of the second operand from the first element of the second operand.

5.7.5. **SSE3 SIMD Floating-Point LOAD/MOVE/DUPLICATE Instructions**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVSHDUP</td>
<td>Loads/moves 128-bits; duplicating the second and fourth 32-bit data elements</td>
</tr>
<tr>
<td>MOVSLDUP</td>
<td>Loads/moves 128-bits; duplicating the first and third 32-bit data elements</td>
</tr>
<tr>
<td>MOVDDUP</td>
<td>Loads/moves 64-bits (bits[63-0] if the source is a register) and returns the same 64 bits in both the lower and upper halves of the 128-bit result register; duplicates the 64 bits from the source</td>
</tr>
</tbody>
</table>

5.7.6. **SSE3 Agent Synchronization Instructions**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONITOR</td>
<td>Sets up an address range used to monitor write-back stores</td>
</tr>
<tr>
<td>MWAIT</td>
<td>Enables a logical processor to enter into an optimized state while waiting for a write-back store to the address range set up by the MONITOR instruction</td>
</tr>
</tbody>
</table>

5.8. **SYSTEM INSTRUCTIONS**

The following system instructions are used to control those functions of the processor that are provided to support for operating systems and executives.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LGDT</td>
<td>Load global descriptor table (GDT) register</td>
</tr>
<tr>
<td>SGDT</td>
<td>Store global descriptor table (GDT) register</td>
</tr>
<tr>
<td>LLDT</td>
<td>Load local descriptor table (LDT) register</td>
</tr>
<tr>
<td>SLDT</td>
<td>Store local descriptor table (LDT) register</td>
</tr>
<tr>
<td>LTR</td>
<td>Load task register</td>
</tr>
<tr>
<td>STR</td>
<td>Store task register</td>
</tr>
<tr>
<td>LIDT</td>
<td>Load interrupt descriptor table (IDT) register</td>
</tr>
<tr>
<td>SIDT</td>
<td>Store interrupt descriptor table (IDT) register</td>
</tr>
<tr>
<td>MOV</td>
<td>Load and store control registers</td>
</tr>
<tr>
<td>LMSW</td>
<td>Load machine status word</td>
</tr>
<tr>
<td>Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>SMSW</td>
<td>Store machine status word</td>
</tr>
<tr>
<td>CLTS</td>
<td>Clear the task-switched flag</td>
</tr>
<tr>
<td>ARPL</td>
<td>Adjust requested privilege level</td>
</tr>
<tr>
<td>LAR</td>
<td>Load access rights</td>
</tr>
<tr>
<td>LSL</td>
<td>Load segment limit</td>
</tr>
<tr>
<td>VERR</td>
<td>Verify segment for reading</td>
</tr>
<tr>
<td>VERW</td>
<td>Verify segment for writing</td>
</tr>
<tr>
<td>MOV</td>
<td>Load and store debug registers</td>
</tr>
<tr>
<td>INVD</td>
<td>Invalidate cache, no writeback</td>
</tr>
<tr>
<td>WBINVD</td>
<td>Invalidate cache, with writeback</td>
</tr>
<tr>
<td>INVLPG</td>
<td>Invalidate TLB Entry</td>
</tr>
<tr>
<td>LOCK (prefix)</td>
<td>Lock Bus</td>
</tr>
<tr>
<td>HLT</td>
<td>Halt processor</td>
</tr>
<tr>
<td>RSM</td>
<td>Return from system management mode (SMM)</td>
</tr>
<tr>
<td>RDMSR</td>
<td>Read model-specific register</td>
</tr>
<tr>
<td>WRMSR</td>
<td>Write model-specific register</td>
</tr>
<tr>
<td>RDPMC</td>
<td>Read performance monitoring counters</td>
</tr>
<tr>
<td>RDTSC</td>
<td>Read time stamp counter</td>
</tr>
<tr>
<td>SYSENER</td>
<td>Fast System Call, transfers to a flat protected mode kernel at CPL=0</td>
</tr>
<tr>
<td>SYSEXIT</td>
<td>Fast System Call, transfers to a flat protected mode kernel at CPL=3</td>
</tr>
</tbody>
</table>
6

Procedure Calls, Interrupts, and Exceptions