IA-32 어셈블리 명령어 모음

1. GENERAL-PURPOSE INSTRUCTIONS ; All IA-32 processors

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 IA32 processors. They operate on data contained in memory, in the general-purpose register(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 regiters (CS, DX, SS, ES, FS, and GS). This group of instructions includes the following subgroups: 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.

1.1 Data Transfer Instructions

The data transfer instructions move data between memory and the general-purpose and segment registers. They also preform 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 register; 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 now below or equal
CMOVAE/CMOVNB - Conditional move if above or equal/Conditional move if not below
CMOVB/CMOVNAE - Conditional move if below/Conditional move if not above or equal
CMOVBE/CMOVNA - Conditional move if below or equal/Conditional move if not above
CMOVG/CMOVNLE - Conditional move if greater/Conditional move if not less or equal
CMOVGE/CMOVNL - Conditional move if greater or equal/Conditional move if not less
CMOVL/CMOVNGE - Conditional move if less/Conditional move if not greater or equal
CMOVLE/CMOVNG - Conditional move if less or equal/Conditional move if not greater
CMOVC - Conditional move if carry
CMOVNC - Conditional move if not carry
CMOVO - Conditional move if overflow
CMOVNO - Conditional move if not overflow
CMOVS - Conditional move if sign(negative)
CMOVNS - Conditional move if not sign(non-negative)
CMOVP/CMOVPE - Conditional move if parity/Conditional move if parity even
CMOVNP/CMOVPO - Conditional move if not parity/Conditional move if parity odd
XCHG - Exchange
BSWAP - Byte swap
XADD - Exchange and add
CMPXCHG - Compare and exchange
CMPXCHG8B - Compare and exchange 8 bytes
PUSH - Push onto stack
POP - Pop off of stack
PUSHA/PUSHAD - Push general-purpose registers onto stack
POPA/POPAD - Pop general-pupose registers from stack
CWD/CDQ - Convert word to doubleword/Convert doubleword to quadword
CBW/CWDE - Convert byte to word/Convert word to doubleword in EAX register
MOVSX - Move and sign extend
MOVZX - Move and zero extend

1.2 Binary Arithmetic Instuctions

The binary arithmetic instructions perform basic binary integer computaions 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

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 subtarction
AAM - ASCII adjust after multiplication
AAD - ASCII adjust before division

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
XOR - Perform bitwise logical exclusive OR
NOT - Perform bitwise logical NOT

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

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 - Blt 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/Set byte if not below or equal/Set byte if now carry
SETB/SETNAE/SETC - Set byte if below/Set byte if not above or equal/Set byte if carry
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 now overflow
SETPE/SETP - Set byte if parity even/Set byte if parity
SETPO/SETNP - Set byte if parity odd/Set byte if now parity
TEST - Logical compare

1.7 Control Transfer Instructions

The control transfer instructions provide jump, conditional jump, loop, and call and return operations to control programs 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/JMB - 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/JMG - 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
JS - Jump if sign(negative)
JNS - Jump of 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

1.8 String Instructions

The string instructions operate on strings of buytes, 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
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

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

1.10 Enter and Leave Instructions

These instructions provide machine-language support for procedure calls in block-structured language.

ENTER - High-level procedure entry
LEAVE - High-level procedure exit

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 info flags
PUSHF/PUSHFD - Push EFLAGS onto stack
POPF/POPFD - Pop EFLAGS from stack
STI - Set interrupt flag
CLI - Clear the interrupt flag

1.12 Segment Register Instructions

The segment register instruction 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

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 adress

NOP - No operation
UD2 - Undefined instruction
XLAT/XLATB - Table lookup translation
CPUID - Processor Identification

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

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-poing value and pop
FILD - Load integer
FIST - Store integer
FISTP - Store integer and pop (*SSE3 provides an instruction FISTTP for integer conversion*)
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

2.2 x87 FPU Basic Arithmetic Instuctions

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
FSUB - Subtract floating-point
FSUBP - Subtract floating-point and pop
FISUB - Subtract integer
FSUBR - Subtract floating-point reverse
FSUBRP - Subtract floating-point reverse and pop
FISUBR - Subtract integer reverse
FMUL - Multiply floating-point
FMULP - Multiplu floating-point and pop
FDIV - Divide floating-point
FDIVP - Divide floating-point and pop
FIDIV - Divide integer
FDIVR - Divide floating-point reverse
FDIVRP - Divide floating-point reverse and pop
FIDIVR - Divide integer reverse
FPREM - Partial remainder
FPREM1 - IEEE Partial remainder
FABS - Absolute value

FCHS - Change sign
FRNDINT - Round to integer
FSCALE - Scale bu power of two
FSQRT - Square root
FXTRACT - Extract exponent and significand

2.3 x87 FPU Comparision Instructions

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

FCOM - Compare floating-point
FCOMP - Compart floating-point and pop
FCOMPP - Compart floating-point and pop twice
FUCOM - Unordered compare floating-point
FUCOMP - Unordered compare floating-point and pop
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

2.4 x87 FPU Transcendental Instructions

The transcendental instructions perform basic trigonometric and logarithmic operations on floating-point operands.

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

2.5 x87 FPU Load Constants Instructions

The load constants instructions load common constants, such as *pie*, into the x87 floating-point registers.

FLD1 - Load +1.0
FLDZ - Load +0.0
FLDPI - Load *pie*
FLDL2E - Load *log*2 e
FLDLN2 - Load *log*e 2
FLDL2T - Load *log*2 10
FLDLG2 - Load *log*10 2

2.6 x87 FPU Control Instructions

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

FINCSTP - Increment FPU register stack pointer
FDECSTP - Decrement FPU fegister stack pointer
FFREE - Free floating-point register
FINIT - Initialize FPU after checking error conditions
FNINIT - Initialize FPU without checking error conditiions
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 CPU 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

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
FXRTOR - Restore x87 Fpu and SIMD state

Initially, these instruction opreated 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 precessor family, these instructions were expanded to also save and restore the state of the XMM and MXCSR registers

4. MMX INSTRUCTIONS

Four extensions have been introduced into the IA-32 architecture to permit IA-32 processors to perform sigle-instruction multiple-data (SIMD) operations. These extensions include the MMX technology, SSE extensions, SSE2 extensions, and SSE3 extensions.
MMX instructions are divided into the following subgroups: datatransfer, conversion, packed arithmetic, comparison, logical, shift and rotate, and state management instruction. The sections that follow introduce each subgroup.

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

4.2 MMX Conversion Instructions

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

PACKSSWB - Pack words into byte with signed saturation
PACKSSDW - Pack doublewords into words with signed saturation
PACKUSWB - Pack words into byte 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

4.3 MMX Packed Arithmetic Instructions

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

PADDB – Add packed byte integers
PADDW – Add packed word integers
PADDD – Add packed doubleword integers
PADDSB – Add packed signed byte integers with signed saturation
PADDSW – 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
PSUBB – Subtract packed byte integers
PSUBW – Subtract packed word integers
PSUBD – Subtract packed doubleword integers
PSUBSB – Subtract packed byte integers with signed saturation
PSUBSW – Subtract packed 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
PMULLW – Multiply packed signed word integers and store low result
PMADDWD – Multiply and add packed word integers

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 doubleword 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

4.5 MMX Logical Instructions

The logical instructions perform AND, AND NOT, OR, and XOR operations on quadword operands.

PAND – Bitwise logical
PANDN – Bitwise logical AND NOT
POR – Bitwise logical OR
PXOR – Bitwise logical exclusive OR

4.6 MMX Shift and Rotate Instructions

The shift and rotate instructions shift and rotate packed butes, word, or doublewords, or quadword 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 rigjt logical
PSRAW – Shift packed words right arithmetic
PSRAD – Shift packed doublewords right arithmetic

4.7 MMX State Management Instructions

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

EMMS – Empty MMX state

5. SSE INSTRUCTIONS

SSE instructions represent an extension of the SIMD execution model introduced with the MMX technology.

SSE instructions can only be executed on IA-32 precessors that support SSE extensions. Support for these instructions can be detected with the CPUID instruction.
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.

5.1 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 subgroup: data transfer, packed arithmetic, comparison, logical, shuffle and unpack, and conversion instructions.

5.1.1 SSE Data Transfer Instructions

SSE data transfer instruction move packed and scalar single-precision floating-point operands between XMM registers and betweeb 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 calues to an from the high quadword of an XMM register and memory
MOVHLPS - Move two packed single-precision floating-point calues to an from the high quadword of an XMM register to the low quadword of another XMM register
MOVLPS - Move two packed single-precision floating-point calues to an from the low quadword of an XMM register and memory.
MOVLHPS - Move two packed single-precision floating-point calues to an from the low quadword of an XMM register to the high quadword of another XMM register
MOVMSKPS - Extract sign mask from four packed single-precision floating-point values
MOVSS - Move scalar single-precision floating-point value between XMM registers or beween an XMM register and memory

5.1.2 SSE Packed Arithmetic Instructions

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

ADDPS - Add packed single-precision floating-point values
ADDSS - Add scalar single-precision floating-point values
SUBPS - Subtract packed single-precision floating-point values
SUBSS - Subtract scalar single-precision floating-point values
MULPS - Multiply packed single-precision floating-point values
MULSS - Multiply scalar single-precision floating-point values
DIVPS - Divide packed single-precision floating-point values
DIVSS - Divide scalar single-precision floating-point values
RCPPS - Compute reciprocals of packed single-precision floating-point values
RCPSS - Compute reciprocals of scalar single-precision floating-point values
SQRTPS - Compute square roots of packed single-precision floating-point values
SQRTSS - Compute square roots of scalar single-precision floating-point values
RSQRTPS - Compute reciprocals of square roots of packed single-precision floating-point values
RSQRTSS - Compute reciprocals of square roots of scalar single-precision floating-point values
MAXPS - Return maximum packed single-precision floating-point values
MAXSS - Return maximum scalar single-precision floating-point values
MINPS - Return minimum packed single-precision floating-point values
MINSS - Return minimum scalar single-precision floating-point values

5.1.3 SSE Comparison Instructions

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

CMPPS - Compare packed single-precision floating-point values
CMPSS - Compare scalar single-precision floating-point values
COMISS - Preform ordered comparison of scalar single-precision floating-point values and set flags in EFLAGS register
UCOMISS - Preform unordered comparison of scalar single-precision floating-point values and set flags in EFLAGS register

5.1.4 SSE Logical Instructions

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

ANDPS - Perform bitwise logical AND of packed single-precision floating-point values
ANDNPS - Perform bitwise logical AND NOT of packed single-precision floating-point values
ORPS - Perform bitwise logical OR of packed single-precision floating-point values
XORPS - Perform bitwise logical XOR of packed single-precision floating-point values

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.

SHUFPS - Shuffles values in packed single-precision floating-point operands
UNPCKHPS - Unpacks and interleaves the two high-order values from two single-precision floating-point operands
UNPCKLPS - Unpacks and interleaves the two low-order values from two single-precision floating-point operands

5.1.6 SSE Conversion Instructions

SSE conversion instructions convert packed and individual double word integers into packed and calar 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 calar single-precision floating-point value
CVTPS2PI - Convert packed single-precision floating-point values to packed doubleword integers
CVTTP2PI - 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 interger
CVTTSS2SI - Convert with truncation a scalar single-precision floating-point value to a scalar doubleword integer

5.2 SSE MXCSR State Management Instructions

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

LDMXCSR - Load MXCSR register
STMXCSR - Save MXCSR register state

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. The represent enhancements to the MMX instruction set described in *"Section 4. MMX Instructions"*.

PAVGB - Compute average of packed unsigned byte integers
PAVGW - Compute average of packed unsigned word integers
PEXTRW - Extract word
PINSRW - Insert word
PMAXUB - Maximum of packed unsigned byte integers
PMAXSW - Maximum of packed signed word integers
PMINUB - Minimum of packed unsigned byte integers

PMINSW - Minimum of packed signed word integers
PMOVMSKB - Move byte mask
PMULHUW - Multiply packed unsigned integers and store high result
PSADBW - Compute sun of absolute differences
PSHUFW - Shuffle packed intger word in MMX register

5.4 SSE Cacheability Control, Prefetch, and Instruction Ordering Instructions

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

MASKMOVQ - Non-temporal store of selected butes 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-precsion floating-point values from an XMMS register info memory
PREFETCH*h* - Load 32 or more of bytes from memory to a selected level of the processor's cache hierarchy
SFENCE - Serializes store operations

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.
SSE2 instructions can only be executed on Intel 64 and IA-32 processors that support the SSE2 extensions. Support for these instructions can be detected with the CPUID instruction.
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
- Cacheability-control and instruction ordering instructions

The following sections give an overview of each subgroup.

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.

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 and XMM register and memory
MOVUPD - Move two unaligned packed double-precision floating-point values between XMM registers or between and 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

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
SQRTSD - 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

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

6.1.4 SSE 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.

CMPPD - Compare packed double-precision floating-point values
CMPSD - 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.

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

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
CVTPI2PD - Convert packed doubleword integers to packed double-precision floating-point values
CVTPD2DQ - Convert packed double-precision floating-point values to packed doubleword integers
CVTTPD2DQ - Convert with truncation packed double-precision floating-point values to packed doubleword integers

CVTDQ2PD – Convert packed doubleword integers to packed double-precision floating-point values
CVTPS2PD – Convert packed single-precision floating-point values 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 doubleword 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

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

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
PUNPCKHQDQ – Unpack high quadwords
PUNPCKLQDQ – Unpack low quadwords

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

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 x87 FPU 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.
The sections that follow describe each subgroup.

7.1 SSE3 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)

7.2 SSE3 Specialized 128-bit Unaligned Data Load Instruction

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

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 quad-words, and double-precision subtraction on the first pair

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.

7.5 SSE3 SIMD Floating-Point LOAD/MOVE/DUPLICATE Instructions

MOVSHDUP - Loads/moves 128 bits; duplicating the second and fourth 32-bit data elements
MOVSLDUP - Loads/moves 128 bits; duplicating the first and third 32-bit data elements
MOVDDUP - 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

7.6 SSE3 Agent Synchronization Instructions

MONITOR - Sets up an address range used to monitor write-back stores
MWAIT - 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

8. SUPPLEMENTAL STREAMING SIMD EXTENSIONS 3 (SSSE3) INSTRUCTIONS

SSSE3 provide 32 instructions (represented by 14 mnemonics) to accelerate computations on packed integers. These include:

- Twelve instructions that perform horizontal addition or subtraction operations.
- Six instructions that evaluate absolute values.
- Two instructions that perform multiply and add operations and speed up the evaluation of dot products.
- Two instructions that accelerate packed-integer multiply operations and produce integer values with scaling.
- Two instructions that perform a byte-wise, in-place shuffle according to the second shuffle control operand.
- Six instructions that negate packed integers in the destination operand if the signs of the corresponding element in the source operand is less than zero.
- Two instructions that align data from the composite of two operands.

SSSE3 instructions can only be executed on IA-32 processors that support SSSE3 extensions. Support for these instructions can be detected with the CPUID instruction.
The sections that follow describe each subgroup.

8.1 Horizontal Addition/Subtraction

PHADDW - Adds two adjacent, signed 16-bit integers horizontally from the source and destination operands and packs the signed 16-bit results to the destination operand.
PHADDSW - Adds two adjacent, signed 16-bit integers horizontally from the source and destination operands and packs the signed, saturated 16-bit results to the destination operand.
PHADDD - Adds two adjacent, signed 32-bit integers horizontally from the source and destination operands and packs the signed 32-bit results to the destination operand.
PHSUBW - Performs horizontal subtraction on each adjacent pair of 16-bit signed integers by subtracting the most significant word from the least significant word of each pair in the source and destination operands. The signed 16-bit results are packed and written to the destination operand.
PHSUBSW - Performs horizontal subtraction on each adjacent pair of 16-bit signed integers by subtracting the most significant word from the least significant word of each pair in the source and destination operands. The signed, saturated 16-bit results are packed and written to the destination operand.
PHSUBD - Performs horizontal subtraction on each adjacent pair of 32-bit signed integers by subtracting the most significant doubleword from the least significant double word of each pair in the source and destination operands. The signed 32-bit results are packed and written to the destination operand.

8.2 Packed Absolute Values

PABSB - Computes the absolute value of each signed byte data element.
PABSW - Computes the absolute value of each signed 16-bit data element.
PABSD - Computes the absolute value of each signed 32-bit data element.

8.3 Multiply and Add Packed Signed and Unsigned Bytes

PMADDUBSW - Multiplies each unsigned byte value with the corresponding signed byte value to produce an intermediate, 16-bit signed integer. Each adjacent pair of 16-bit signed values are added horizontally. The signed, saturated 16-bit results are packed to the destination operand.

8.4 Packed Multiply High with Round and Scale

PMULHRSW - Multiplies vertically each signed 16-bit integer from the destination operand with the corresponding signed 16-bit integer of the source operand, producing intermediate, signed 32-bit integers. Each intermediate 32-bit integer is truncated to the 18 most significant bits. Rounding is always performed by adding 1 to the least significant bit of the 18-bit intermediate result. The final result is obtained by selecting the 16 bits immediately to the right of the most significant bit of each 18-bit intermediate result and packed to the destination operand.

8.5 Packed Shuffle Bytes

PSHUFB - Permutes each byte in place, according to a shuffle control mask. The least significant three or four bits of each shuffle control byte of the control mask form the shuffle

index. The shuffle mask is unaffected. If the most significant bit (bit 7) of a shuffle control byte is set, the constant zero is written in the result byte.

8.6 Packed Sign

PSIGNB/W/D – Negates each signed integer element of the destination operand if the sign of the corresponding data element in the source operand is less than zero.

8.7 Packed Align Right

PALIGNR – Source operand is appended after the destination operand forming an intermediate value of twice the width of an operand. The result is extracted from the intermediate value into the destination operand by selecting the 128 bit or 64 bit value that are right-aligned to the byte offset specified by the immediate value.

9 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.

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

10. 64-BIT MODE INSTRUCTIONS

The following instructions are introduced in 64-bit mode. This mode is a sub-mode of IA-32e mode.

CDQE – Convert doubleword to quadword
CMPSQ – Compare string operands
CMPXCHG16B – Compare RDX:RAX with m128
LODSQ – Load qword at address (R)SI into RAX
MOVSQ – Move qword from address (R)SI to (R)DI
MOVZX (64-bits) – Move doubleword to quadword, zero-extension
STOSQ – Store RAX at address RDI

SWAPGS - Exchanges current GS base register value with value in MSR address C0000102H
SYSCALL - Fast call to privilege level 0 system procedures
SYSRET - Return from fast system call

11. VIRTUAL-MACHINE EXTENSIONS

The behavior of the VMCS-maintenance instructions is summarized below:

VMPTRLD - Takes a single 64-bit source operand in memory. It makes the referenced VMCS active and current.
VMPTRST - Takes a single 64-bit destination operand that is in memory. Current-VMCS pointer is stored into the destination operand.
VMCLEAR - Takes a single 64-bit operand in memory. The instruction sets the launch state of the VMCS referenced by the operand to "clear", renders that VMCS inactive, and ensures that data for the VMCS have been written to the VMCS-data area in the referenced VMCS region.
VMREAD - Reads a component from the VMCS (the encoding of that field is given in a register operand) and stores it into a destination operand.
VMWRITE - Writes a component to the VMCS (the encoding of that field is given in a register operand) from a source operand.

The behavior of the VMX management instructions is summarized below:

VMCALL - Allows a guest in VMX non-root operation to call the VMM for service. A VM exit occurs, transferring control to the VMM.
VMLAUNCH - Launches a virtual machine managed by the VMCS. A VM entry occurs, transferring control to the VM.
VMRESUME - Resumes a virtual machine managed by the VMCS. A VM entry occurs, transferring control to the VM.
VMXOFF - Causes the processor to leave VMX operation.
VMXON - Takes a single 64-bit source operand in memory. It causes a logical processor to enter VMX root operation and to use the memory referenced by the operand to support VMX operation.