

## 5.1. GENERAL-PURPOSE INSTRUCTIONS

The general-purpose instructions perform 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 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.

### 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
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-purpose registers from stack
IN	Read from a port
OUT	Write to a port
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

### 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

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
XOR	Perform bitwise logical exclusive OR
NOT	Perform bitwise logical NOT

### 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

The bit and instructions test and modify individual bits in the bits in word and doubleword operands. The 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
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
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/MOVS	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
INS/INSB	Input string from port/Input byte string from port

## INSTRUCTION SET SUMMARY



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

### 5.1.9. Flag Control 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

### 5.1.10. 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



### 5.1.11. 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.

### 5.2.1. Data Transfer

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