| Type | Form | Operand value | Name |
|---|---|---|---|
| Immediate | $\$Imm$ | $Imm$ | Immediate |
| Register | $\mathtt{E}_a$ | $\mathtt{R}[\mathtt{E}_a]$ | Register |
| Memory | $Imm$ | $\mathtt{M}[Imm]$ | Absolute |
| Memory | $(\mathtt{E}_a)$ | $\mathtt{M}[\mathtt{R}[\mathtt{E}_a]]$ | Indirect |
| Memory | $Imm(\mathtt{E}_b)$ | $\mathtt{M}[Imm + \mathtt{R}[\mathtt{E}_b]]$ | Base + displacement |
| Memory | $(\mathtt{E}_b, \mathtt{E}_i)$ | $\mathtt{M}[\mathtt{R}[\mathtt{E}_b] + \mathtt{R}[\mathtt{E}_i]]$ | Indexed |
| Memory | $Imm(\mathtt{E}_b, \mathtt{E}_i)$ | $\mathtt{M}[Imm + \mathtt{R}[\mathtt{E}_b] + \mathtt{R}[\mathtt{E}_i]]$ | Indexed |
| Memory | $(, \mathtt{E}_i, s)$ | $\mathtt{M}[\mathtt{R}[\mathtt{E}_i] \cdot s]$ | Scaled indexed |
| Memory | $Imm(, \mathtt{E}_i, s)$ | $\mathtt{M}[Imm + \mathtt{R}[\mathtt{E}_i] \cdot s]$ | Scaled indexed |
| Memory | $(\mathtt{E}_b, \mathtt{E}_i, s)$ | $\mathtt{M}[\mathtt{R}[\mathtt{E}_b] + \mathtt{R}[\mathtt{E}_i] \cdot s]$ | Scaled indexed |
| Memory | $Imm(\mathtt{E}_b, \mathtt{E}_i, s)$ | $\mathtt{M}[Imm + \mathtt{R}[\mathtt{E}_b] + \mathtt{R}[\mathtt{E}_i] \cdot s]$ | Scaled indexed |

Figure 3.3 **Operand forms.** Operands can denote immediate (constant) values, register values, or values from memory. The scaling factor $s$ must be either 1, 2, 4, or 8.

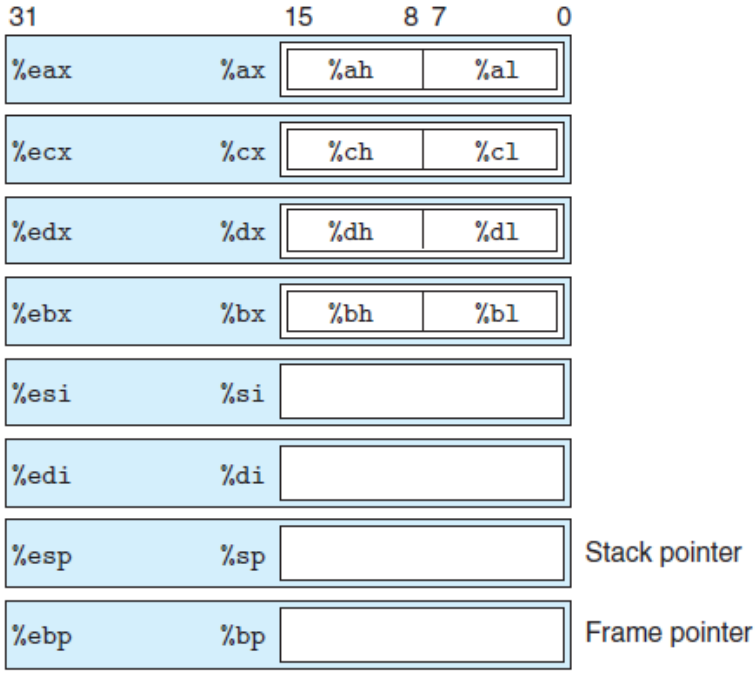
| Instruction | | Effect | Description |
|---|---|---|---|
| MOV | $S$, $D$ | $D \leftarrow S$ | Move |
| `movb` | | Move byte | |
| `movw` | | Move word | |
| `movl` | | Move double word | |
| MOVS | $S$, $D$ | $D \leftarrow \mathrm{SignExtend}(S)$ | Move with sign extension |
| `movsbw` | | Move sign-extended byte to word | |
| `movsbl` | | Move sign-extended byte to double word | |
| `movswl` | | Move sign-extended word to double word | |
| MOVZ | $S$, $D$ | $D \leftarrow \mathrm{ZeroExtend}(S)$ | Move with zero extension |
| `movzbw` | | Move zero-extended byte to word | |
| `movzbl` | | Move zero-extended byte to double word | |
| `movzwl` | | Move zero-extended word to double word | |
| `pushl` | $S$ | $\mathtt{R}[\%\mathtt{esp}] \leftarrow \mathtt{R}[\%\mathtt{esp}] - 4$; $\mathtt{M}[\mathtt{R}[\%\mathtt{esp}]] \leftarrow S$ | Push double word |
| `popl` | $D$ | $D \leftarrow \mathtt{M}[\mathtt{R}[\%\mathtt{esp}]]$; $\mathtt{R}[\%\mathtt{esp}] \leftarrow \mathtt{R}[\%\mathtt{esp}] + 4$ | Pop double word |

Figure 3.4 **Data movement instructions.**

| Instruction | | Effect | Description |
|---|---|---|---|
| `leal` | $S, D$ | $D \leftarrow \&S$ | Load effective address |
| INC | $D$ | $D \leftarrow D + 1$ | Increment |
| DEC | $D$ | $D \leftarrow D - 1$ | Decrement |
| NEG | $D$ | $D \leftarrow -D$ | Negate |
| NOT | $D$ | $D \leftarrow \sim D$ | Complement |
| ADD | $S, D$ | $D \leftarrow D + S$ | Add |
| SUB | $S, D$ | $D \leftarrow D - S$ | Subtract |
| IMUL | $S, D$ | $D \leftarrow D * S$ | Multiply |
| XOR | $S, D$ | $D \leftarrow D \hat{} S$ | Exclusive-or |
| OR | $S, D$ | $D \leftarrow D \mid S$ | Or |
| AND | $S, D$ | $D \leftarrow D \& S$ | And |
| SAL | $k, D$ | $D \leftarrow D << k$ | Left shift |
| SHL | $k, D$ | $D \leftarrow D << k$ | Left shift (same as SAL) |
| SAR | $k, D$ | $D \leftarrow D >>_A k$ | Arithmetic right shift |
| SHR | $k, D$ | $D \leftarrow D >>_L k$ | Logical right shift |

Figure 3.7 **Integer arithmetic operations.** The load effective address (`leal`) instruction is commonly used to perform simple arithmetic. The remaining ones are more standard unary or binary operations. We use the notation $>>_A$ and $>>_L$ to denote arithmetic and logical right shift, respectively. Note the nonintuitive ordering of the operands with ATT-format assembly code.

Figure 3.2 **IA32 integer registers.** All eight registers can be accessed as either 16 bits (word) or 32 bits (double word). The 2 low-order bytes of the first four registers can be accessed independently.



| Instruction | | Based on | Description |
|---|---|---|---|
| CMP | $S_2$, $S_1$ | $S_1 - S_2$ | Compare |
| `cmpb` | | Compare byte | |
| `cmpw` | | Compare word | |
| `cmpl` | | Compare double word | |
| TEST | $S_2$, $S_1$ | $S_1$ & $S_2$ | Test |
| `testb` | | Test byte | |
| `testw` | | Test word | |
| `testl` | | Test double word | |

Figure 3.10 **Comparison and test instructions.** These instructions set the condition codes without updating any other registers.

| Instruction | | Synonym | Jump condition | Description |
|---|---|---|---|---|
| `jmp` | *Label* | | 1 | Direct jump |
| `jmp` | **Operand* | | 1 | Indirect jump |
| `je` | *Label* | `jz` | `ZF` | Equal / zero |
| `jne` | *Label* | `jnz` | `~ZF` | Not equal / not zero |
| `js` | *Label* | | `SF` | Negative |
| `jns` | *Label* | | `~SF` | Nonnegative |
| `jg` | *Label* | `jnle` | `~(SF ^ OF) & ~ZF` | Greater (signed >) |
| `jge` | *Label* | `jnl` | `~(SF ^ OF)` | Greater or equal (signed >=) |
| `jl` | *Label* | `jnge` | `SF ^ OF` | Less (signed <) |
| `jle` | *Label* | `jng` | `(SF ^ OF) \| ZF` | Less or equal (signed <=) |
| `ja` | *Label* | `jnbe` | `~CF & ~ZF` | Above (unsigned >) |
| `jae` | *Label* | `jnb` | `~CF` | Above or equal (unsigned >=) |
| `jb` | *Label* | `jnae` | `CF` | Below (unsigned <) |
| `jbe` | *Label* | `jna` | `CF \| ZF` | Below or equal (unsigned <=) |

Figure 3.12 **The jump instructions.** These instructions jump to a labeled destination when the jump condition holds. Some instructions have "synonyms," alternate names for the same machine instruction.

`CF`: Carry Flag. The most recent operation generated a carry out of the most significant bit. Used to detect overflow for unsigned operations.

`ZF`: Zero Flag. The most recent operation yielded zero.

`SF`: Sign Flag. The most recent operation yielded a negative value.

`OF`: Overflow Flag. The most recent operation caused a two's-complement overflow—either negative or positive.