# CS100
# IA32 Assembly Instructions

**Registers**
%eax, %ebx, %ecx, %edx, %esi, %edi, %esp, %ebp

**Operand Forms (for S & D)**

| Type | Form | Value |
|---|---|---|
| Immediate | $Imm | Imm |
| Register | Ea | R[Ea] |
| Memory | Imm | M[Imm] |
| Memory | (Ea) | M[R[Ea]] |
| Memory | Imm(Eb) | M[Imm + R[Eb]] |
| Memory | (Eb,Ei) | M[R[Eb] + R[Ei]] |
| Memory | Imm(Eb,Ei) | M[Imm+R[Eb] + R[Ei]] |
| Memory | ( , Ei , s) | M[R[Ei]*s] |
| Memory | Imm( , Ei, s) | M[Imm + R[Ei]*s] |
| Memory | (Eb, Ei, s) | M[R[Eb] + R[Ei]*s] |
| Memory | Imm(Eb, Ei, s) | M[Imm+R[Eb]+R[Ei]*s] |

**Instructions specify size of operand with a suffix for byte (b), 16-bit (w), or 32-bit (l)**

**Data Movement Instructions (only one memory operand)**

| | | |
|---|---|---|
| mov | S, D | D ⇐ S Move |
| movs | S, D | D ⇐ S Move sign-extended |
| movz | S, D | D ⇐ S Move zero-extended |
| pushl | S | push double word (32-bits) |
| popl | D | pop double word (32-bits) |

**Arithmetic & Logical Instructions**

| | | |
|---|---|---|
| leal | S, D | D ⇐ &S Load effective address |
| inc | D | increment |
| dec | D | decrement |
| neg | D | negate |
| not | D | complement |
| add | S, D | add |
| sub | S, D | subtract |
| imul | S, D | multiply |
| xor | S, D | exclusive-or |
| or | S, D | or |
| and | S, D | and |
| sal | k, D | left shift |
| shl | k, D | left shift |
| sar | k, D | arithmetic right shift |
| shr | k, D | logical right shift |
| imull | S | |
| mull | S | |
| cltd | S | |
| idivl | S | |
| divl | S | |

**Comparison Instructions**

| | | |
|---|---|---|
| cmp | S2, S1 | S1 - S2 |
| test | S2, S1 | S1 & S2 |

| | | |
|---|---|---|
| sete | D | set equal |
| setne | D | set noequal |
| sets | D | D <- sign flag |
| setns | D | D <- ~sign flag |
| setg | D | set greater |
| setge | D | set greater or equal |
| setl | D | set less |
| setle | D | set less or equal |
| seta | D | set above |
| setae | D | set above or equal |
| setb | D | set below |
| setbe | D | set below or equal |

**Branch and Jump Instructions**

| | | |
|---|---|---|
| jmp | *Label* | direct jump |
| jmp | *Operand | indirect jump |
| je | *Label* | equal/zero |
| jne | *Label* | not equal/ not zero |
| js | *Label* | negative |
| jns | *Label* | nonnegative |
| jg | *Label* | greater |
| jge | *Label* | greater or equal |
| jl | *Label* | less |
| jle | *Label* | less or equal |
| ja | *Label* | above |
| jae | *Label* | above or equal |
| jb | *Label* | below |
| jbe | *Label* | below or equal |

**Assembly Syntax**

**.align n**
Align the next datum on a 2 n byte boundary. For example, .align 2 aligns the next value on a word boundary. .align 0 turns off automatic alignment of .half, .word, .float, and .double directives until the next .data or .kdata directive.

**.ascii str**
Store the string in memory, but do not null-terminate it.

**.asciz str**
Store the string in memory and null-terminate it.

**.byte b1, ..., bn**
Store the n values in successive bytes of memory.

**.data <addr>**
The following data items should be stored in the data segment. If the optional argument addr is present, the items are stored beginning at address addr .

**.double d1, ..., dn**
Store the n floating point double precision numbers in successive memory locations.

**.extern sym size**
Declare that the datum stored at sym is size bytes large and is a global symbol. This directive enables the assembler to store the datum in a portion of the data segment that is efficiently accessed via register $gp.

**.float f1, ..., fn**
Store the n floating point single precision numbers in successive memory locations.

**.globl sym**
Declare that symbol sym is global and can be referenced from other files.

**.half h1, ..., hn**
Store the n 16-bit quantities in successive memory halfwords.

**.space n**
Allocate n bytes of space in the current segment (which must be the data segment in SPIM).

**.text <addr>**
The next items are put in the user text segment. In SPIM, these items may only be instructions or words (see the .word directive below). If the optional argument addr is present, the items are stored beginning at address addr .

**.word w1, ..., wn**
Store the n 32-bit quantities in successive memory words.