# x86 cheat sheet

### general purpose registers

```
%eax    (%ax,%ah,%al)
%ecx    (%cx,%ch,%cl)
%edx    (%dx,%dh,%dl)
%ebx    (%bx,%bh,%bl)
%esi
%edi
%ebp [base pointer]
%esp [stack pointer]
```

### program counter

```
%eip
[instruction pointer]
```

### condition codes (CCs)

```
cf (carry flag)
zf (zero flag)
sf (sign flag)
of (overflowing flag)
```

### jump

| | |
|---|---|
| `j dst` | always jump |
| `je dst` | jump if equal/zero |
| `jne dst` | … not eq/not zero |
| `js dst` | … negative |
| `jns dst` | … non-negative |
| `jg dst` | … greater (signed) |
| `jge dst` | … >= (signed) |
| `jl dst` | … less (signed) |
| `jle dst` | … <= (signed) |
| `ja dst` | … above (unsigned) |
| `jb dst` | … below (unsigned) |

dst is address of code (i.e., jump target)

### comparison

```
cmpl src2, src1
```

// like computing src1 - src2

```
cf=1 if carry out from msb
zf=1 if (src1==src2)
sf=1 if (src1-src2 < 0)
of=1 if two's complement
        under/overflow
```

### testing

```
testl src2, src1
```

// like computing src1 & src2

```
zf set when src1&src2 == 0
sf set when src1&src2 < 0
```

### set

| | |
|---|---|
| `sete dst` | equal/zero |
| `setne dst` | not eq/not zero |
| `sets dst` | negative |
| `setns dst` | non-negative |
| `setg dst` | greater (signed) |
| `setge dst` | >= (signed) |
| `setl dst` | less (signed) |
| `setle dst` | <= (signed) |
| `seta dst` | above (unsigned) |
| `setb dst` | below (unsigned) |

`dst` must be one of the 8 single-byte reg (e.g., %al)

often paired with `movzbl` instruction
(which moves 8-byte reg into 32-bit & zeroes out rest)

### data movement

```
movl src, dst
```

src or dot can be:
- immediate (e.g., $0x10 or $4)
- register (e.g., %eax)
- memory (e.g., an address)

limits:
- dst can never be an immediate
- src or dot (but not both) can be memory

**general memory form:**

```
N (register1, register2, C)
```

which leads to the memory address:

```
N + register1 + (C * register2)
```

N can be a large number;
C can be 1, 2, 4, or 8

**common shorter forms:**

| | |
|---|---|
| `N` | absolute (reg1=0,reg2=0) |
| `(%eax)` | register indirect (N=0,reg2=0) |
| `N(%eax)` | base + displacement (reg2=0) |
| `N(%eax,%ebx)` | indexed (C=1) |

**example:**

```
movl 4(%eax), %ebx
```

takes value inside register %eax, adds 4 to it, and then fetches the contents of memory at that address, putting the result into register %ebx; sometimes called a "load" instruction as it loads data from memory into a register

### arithmetic

**two operand instructions**

| | |
|---|---|
| `addl src,dst` | `dst = dst + src` |
| `subl src,dst` | `dst = dst - src` |
| `imull src,dst` | `dst = dst * src` |
| `sall src,dst` | `dst = dst << src (aka shll)` |
| `sarl src,dst` | `dst = dst >> src (arith)` |
| `shrl src,dst` | `dst = dst >> src (logical)` |
| `xorl src,dst` | `dst = dst ^ src` |
| `andl src,dst` | `dst = dst & src` |
| `orl src,dst` | `dst = dst \| src` |

**one operand instructions**

| | |
|---|---|
| `incl dst` | `dst = dst + 1` |
| `decl dst` | `dst = dst - 1` |
| `negl dst` | `dst = -dst` |
| `notl dst` | `dst = ~dst` |

**arithmetic ops set CCs implicitly**

```
cf=1 if carry out from msb
zf=1 if dst==0,
sf=1 if dst < 0 (signed)
of=1 if two's complement
     (signed) under/overflow
```