Addressing and Basic Machine Language

On 32-bit systems (x86) logical address space is 64 GB

\[ 2^{32} = 2^6 \times 2^{26} \]
\[ \approx 64 \times 10^9 \]

Segment offset addressing
36 bits of address on 32-bit architecture

"segment" is 4 GB range of memory space
within segment only need 32 bits

Individual segments used for:
- code
- data
- stack

CS, DS, SS 32-bit reg store segment address
bits of reg. reg. are high order bits of reg. addr
how 4 bits were 0.

offset address identify specific mem. location
within segment
these are "adds" manipulated by program

Notation: pair is written segment: offset
eg.
\[ \text{0A7C312E:00000F38} \]
logical address = segment x $2^{16}$ + offset

\[ \begin{array}{c}
0A7C312E \\
00000F38 \\
\hline
0A7C39218 \end{array} \quad \text{logical address}
\]

reg. reg. managed by OS

only worry about offset
Instructions must specify 24 bits for operand

- **Address modes**
  - register addressing - operand is in register
  - immediate addressing - operand is explicitly present

  \[
  \text{add} \quad %%eax, %ebx, %ecx \quad %ecx \leftarrow %ecx - 10
  \]

  \[
  \text{mov} \quad %edx, %%eax \quad %ecx \leftarrow M[%%eax]
  \]

- **Register addressing**
  - operand is explicitly in register

  \[
  \text{add} \quad 0x24, %ecx, %ecx \leftarrow %ecx + M[0x24]
  \]

- **Memory addressing**
  - operand is in memory

  \[
  \text{mov} \quad %edx, %esi \quad M[%%esi] \leftarrow %edx
  \]

- **Immediate offset, address**
  - mem addr of operand is sum of literal (op inst) and register (called offset)

  \[
  \text{mov} \quad %esi, %ecx, %esi
  \]

  \[
  \%ecx \leftarrow M[%%esi + %esi]
  \]

**Intel terms**

- **base** - displacement
- **"base" register** - "displacement" is literal

- **Indirect addressing**
  - memory addr stored in reg
  - register indirect

  \[
  \text{mov} \quad (%%ebx, %esi), %ecx \quad %ecx \leftarrow M[%%ebx]
  \]

  \[
  \text{add} \quad (%%ebx, %esi), %ecx \quad %ecx \leftarrow %ecx + M[%%ebx]
  \]

  \[
  \%eax \leftarrow \%ecx + M[%%ebx + 2*%esi]
  \]
scaled indexed 

movl $0x7f, (%ebx, %edi, 2)
M[0x0b + 0xel + 2] <- 0x7f

full form - literal_direct, base, index, scale
movl 0x50(%ebx, %edi, 4), %eax
%eax <- M[0x50 + %ebx + %edi*4]

---

Basic Prog. Structure

statements + directives
statements translate into machine inst.
directive is not for assembler
addl $01, %eax

<op>l 32-bit operand
<op<w 16-bit operand
<op> 8 8-bit operand

---

standard form

label op_code operands comment

/* comments */

# comment