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

\[ 2^{36} = 2^6 \times 2^{30} \approx 64 \times 10^9 \]

Sig must-offset addressing

36 bits of addr on 32-bit arch
"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 addr
bits of seg. reg. are high order bits of reg. addr.
how 4 bits are 0.
offset addresses identify specific mem. locations within segment

these are "addr" manipulated by program

Notation: pair is written segment: offset

e.g. 0A7C312E: 00007F3F

logical addr: segment \times 2^{16} + offset
\[\begin{align*}
\text{OAFC312E0} & \\
\text{00007F3E} & \\
\text{OAFC3921F} & \leftarrow \text{logical addr.}
\end{align*}\]

reg. reg. managed by OS

only worry about offset
Instructions must specify data for operations.

- Address modes:
  - Register addressing - operand is in register

```
addl %eax, %ebx  %ebx = %ebx + %eax
```

- Operation (op code)
  - Destination
  - Source operand
- Immediate address operand is explicitly present

    subb $10, %edx
    %edx ← %edx - 10

    ^
    designates "immediate" operand

- Direct/absolute address operand is explicitly in inst.

    addl 0x24, %ecx
    %ecx ← %ecx + M[0x24]

    movl %edx, var1
    M[var1] ← %edx
Direct offset addr - mem addr of operand is sum of literal val (in inst) and register (called offset)

movl %var2, %ecx

%ecx <- M[%var2 + %esi]

Intel terms
base - displacement
"base" register "displacement" is literal
Indirect addr memory addr stored in reg. register indirect

movl (%ebx), %eax  \( %\text{eax} \leftarrow M[%\text{ebx}] \)

Indexed addr add two reg to form addr

addl (%ebx, %esi), %eax

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

base \downarrow index
scaled indexed, scale reg by 1, 2, 4, 8

```
movl  $0x7f, (%ebx, %edi, 2)
```

\[
M[\%ebx + \%edi*2] \leftarrow 0x7f
\]

full form - literal (direct), base, index, scale

```
movl  0x50(\%ebx, \%esi, 4), \%eax
```

\[
\%eax \leftarrow M[0x50 + \%ebx + \%esi*4]
\]
BasicProg. Structure

Statements + directives

Statements translate into machine inst.
Directive is inst. for assembler

addl $01, $eax

<op> l  32-bit operands
<op> w  16-bit 
<op> b  8-bit 

standard form

label opcode operands comment

// comments */

' comment

# comment