CS 354 - Lecture 6

1. Registers
   - data transfer
   - arithmetic

2. Instructions

3. Memory
   - X

C program → compiler → Machine code (1010101010)

C → Assembly (human readable mostly)

summ = \( x + y \)

add! OP1, OP2
Assembly

- Intel
- ARM
- SPARC

X86 Assembly

- 8086
- 80186
- 80286
- Pentium

X86

AT&T

(UNIX)

Intel

MS-DOS, Windows

Add source dest

\[ \text{add } A \rightarrow B \]

\[ B \leftarrow A + B \]
Program Counter or Instruction Pointer

Process

fetch

decode

execute

\[ \text{sum} = X + Y \]

Review

CPU

%eip 0x1004

next time, should eip point to next instr.

Memory

code

data

\[ \text{add A, B} \]

1011 0111 1111

assembler
- small regions of memory in a CPU
- each have a name
- read/write to them.
32-bit machines

%eax  →  ax

←  32-bits  →

Int / Pointer

1 byte

a

2 byte

ax  ah  al

Instructions

- Data transfer
- Arithmetic
- Memory
- Control flow (if, functions, while)
Data transfer

MOV source operand, dest. operand

\[
\text{mov } \$21, \%eax
\]

Can't see in x86, immediate value.

\[
\text{mov } \%eax, \%ebx
\]

Variants

\[
\begin{align*}
\text{movl} & \rightarrow \text{long word (4 bytes)} \\
\text{movw} & \rightarrow 2 \text{ bytes (movw } \$12, \%ax) \\
\text{movb} & \rightarrow \text{byte (movb } \$10, \%al) \\
\end{align*}
\]

\[
\text{movl } \$21, \%eax.
\]
Arithmetic instructions

```
addl source, dest
    dest ← source + dest

subl src, dest
    dest ← dest - src

imull src, dest
    dest ← dest * src

alt:
    imull aux, src, dest
    dest ← aux * src

idivl operand
    divisor
```

<table>
<thead>
<tr>
<th>Upper</th>
<th>Lower</th>
</tr>
</thead>
<tbody>
<tr>
<td>%edx</td>
<td>%eax</td>
</tr>
</tbody>
</table>

8 bytes

\[
\frac{9}{2} → Q = 4, R = 1
\]

Quotient → %eax
Remainder → %edx
Memory

1. Direct access (Absolute)
   `movl %eax, 0x1000`
   `movl 0x1004, %ebx`
   `movl $0x1000, %ebx`
   `movl %ebx, 71`

2. Indirect memory access
   `movl (%eax), %ebx`
movl $8(\%eax, \%ebx, 4), \%ecx

![Scaled Index](image)

\[
\text{addr} = 8 + \%eax + (\text{content of } \%ebx \times 4)
\]

\[
\text{movl \ Imm(R1, R2, S), \%R3}
\]

\[
\text{Imm + R1 + S \times R2} \rightarrow \text{memory location}
\]

\[
\text{movl \ 8(\%eax), \%ecx}
\]

\[
\begin{array}{c}
\%eax \\
\text{0x1000}
\end{array}
\]

\[
\begin{array}{c}
\text{0x1000} \\
\text{+0x8}
\end{array}
\]

\[
\begin{array}{c}
\text{0x1008}
\end{array}
\]

\[
\begin{array}{c}
\%ecx \\
\text{21}
\end{array}
\]
movl (%eax, %ebx, 4), %eax
addr = %eax + 4 * %ebx
movl 8 (%ebx, 4), %ecx
addr = 8 + 4 * %ebx
Problem #1
Write assembly to:
- move value 1 into %eax
- add 10 to it and put result into %eax

\[
\begin{align*}
\text{movl} & \quad 1, \%eax \\
\text{addl} & \quad 10, \%eax
\end{align*}
\]

Problem #2
Expression: \(3 + (6 \times 2)\)
Use one register (%eax), and 3 instructions to compute this piece-by-piece

\[
\begin{align*}
\text{movl} & \quad 6, \%eax \\
\text{imull} & \quad 2, \%eax \\
\text{addl} & \quad 3, \%eax
\end{align*}
\]

Problem #3
movl $0, %edx
movl $7, %eax
movl $3, %ebx
idivl %ebx
movl %eax, %ecx
movl $0, %edx
movl $9, %eax
movl $2, %ebx
idivl %ebx
movl %edx, %eax
addl %ecx, %eax

Write simple C expression that is equivalent to these instructions

\[
\begin{align*}
\text{d} & \leftarrow \frac{7}{3} + \frac{9}{2} \\
\end{align*}
\]
Problem #4 (from CSAPP 3.1)

Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100</td>
<td>0xFF</td>
</tr>
<tr>
<td>0x104</td>
<td>0xAB</td>
</tr>
<tr>
<td>0x108</td>
<td>0x13</td>
</tr>
<tr>
<td>0x10C</td>
<td>0x11</td>
</tr>
</tbody>
</table>

Registers

| %eax   | 0x100 |
| %ecx   | 0x1   |
| %edx   | 0x3   |

Value of:

- %eax: 0x100
- 0x104: 0xAB
- $0x108: 0x108
- (%eax): 0xFF
- 4(%eax): 0xAB
- 9(%eax, %edx): 0x11
- 260(%ecx, %edx): 0x13
- 0xFC(%ecx, 4): 0xFF
- (%eax, %edx, 4): 0x11

addr = 4 + 0x100 = 0x104

addr = 9 + 0x100 + 0x3

260 + 0x4 = 264

1

0x100

0x0009

0x10C

0x10C

0x000C

0x10C

0x10D

0x8

264

16