Intro
Study Assembly (Recap project 0, -m32)

Motivation
- why study assembly
- high level, assembly & machine code diff

Goal of the lecture.

```c
int accum = 0;
int sum(int x, int y)
{
    int t = x+y;
    accum += t;
    return t;
}
```

Machine languages
- IA32, x86-64

addressable range of IA32 = 4GB (byte addressable)

Reverse understanding
0010 0001 0010 0011

Add

Memory models
- Flat memory model
- Segmented memory model

```
Seg  offset  Code Seg  Data Seg
```

```
43.07, -89.40
```
Process Registers: Understand Processor state

- Program Counter (PC)
  Address of next instruction to be executed

- Register files
  "named" locations to store 32-bit values like pointers or int data

- Condition Code registers
  Status of most recent arithmetic/logical instruction (e.g., overflow in addition)

- Floating point registers

Data formats

- More restrictive (less options) than C

Gerald's Sheet 2

Integer Registers

- 8 types of registers, sub-registers.
<table>
<thead>
<tr>
<th>Type</th>
<th>Form</th>
<th>Operand Value</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>$Imm$</td>
<td>Imm</td>
<td>Immediate</td>
</tr>
<tr>
<td>Register</td>
<td>R[De]</td>
<td>R[De]</td>
<td>Register</td>
</tr>
<tr>
<td>Memory</td>
<td>M[Imm]</td>
<td>Absolute</td>
<td>Indirect</td>
</tr>
<tr>
<td>–II–</td>
<td>M[R[De]]</td>
<td>Base + displacement</td>
<td>Indexed</td>
</tr>
<tr>
<td>–II–</td>
<td>M[Imm + R[De] + R[Di]]</td>
<td>Scaled indexed</td>
<td></td>
</tr>
<tr>
<td>–II–</td>
<td>M[Imm + R[De] + R[Di] + R[Si]]</td>
<td>Scaled indexed</td>
<td></td>
</tr>
</tbody>
</table>

Note: Scaled indexed format will always have "s" term.

Lecture: Student questions

1. Types of registers. (PC → EIP) (Reg files → EAX, ——) (Condition Code Regs → EFLAGS)
2. A → Accumulator, B → Base C → Counter D → Data
3. How does it even work? C → Assembly w/ so few registers.
4. Why few registers (Power, area limitation)

Review

1. Register AL, AH, AX, EAX.
2. Data formats Byte (1) → b, Word (2) → w, DW (4) → 4
   short
   int
3. Operands Forms.

Lecture

Practice problem 3-1
Practice Problem 3.1

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Register Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100</td>
<td>0xFF</td>
<td>%eax 0x100</td>
</tr>
<tr>
<td>0x104</td>
<td>0xAB</td>
<td>%ecx 0x1</td>
</tr>
<tr>
<td>0x108</td>
<td>0x13</td>
<td>%edx 0x3</td>
</tr>
<tr>
<td>0x10C</td>
<td>0x11</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operand</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td>0x100</td>
</tr>
<tr>
<td>0x104</td>
<td></td>
</tr>
<tr>
<td>$0x108</td>
<td></td>
</tr>
<tr>
<td>(%eax)</td>
<td>0xFF</td>
</tr>
</tbody>
</table>

$9(%eax, %edx) \rightarrow \text{address} = 9 + 0x100 + 0x3 = 9 + 256 + 3 = 268$

$260(%ecx, %edx) \rightarrow \text{address} = 260 + 0x1 + 0x3 = 264 = 0x108$

$0xFC (\%ecx, 4) \rightarrow \text{value} = 0x13.$

$0xFC (\%ecx, 4) \rightarrow \text{address} = 0xFC + 0x1 \times 4 = 0xFC + 4 = 15 \times 16 + 12 + 4 = 256 + 0x100$

$\%eax, \%edx, 4) \rightarrow \text{Address} = 0x100 + 0x3 \times 4 = 0x10C$

$\%eax, \%edx, 4) \rightarrow \text{Value} = 0x11.$
Stack Data Movement

Move to San Diego story.

Stack Operations. Initially

<table>
<thead>
<tr>
<th>Higher address</th>
<th>(bottom)</th>
<th>Pushl %eax</th>
<th>Popl %edx</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax 0x123</td>
<td>%edx 0</td>
<td>%eax 0x123</td>
<td>%edx 0</td>
</tr>
<tr>
<td>%esp 0x108</td>
<td></td>
<td>%esp 0x108</td>
<td></td>
</tr>
</tbody>
</table>

0x108

0x108

0x123

Practice Problem 3.2: Determine Suffix (b/w/l)

mov %eax, %esp. (l)
mov (%eax), %dx (w)
mov $0xFF, %bl (b)
push $0xFF (b)

# Move mem₁ → memory₂ not allowed in 1 instruction

- Move mem₁, reg, move reg, mem₂
- e.g. Gerald pg. 5

Leaf (Gerald pg. 6)

→ has no variation in suffix (why?)
Examples from last lecture

Address = \( O_x100 + O_x3 \times 4 \) = \( O_x100 \)

contents = \( O_x11 \)

\[ \rightarrow \text{explain movb} \ (\%eax), \%dh \]

\[ \text{mov} \ (\%eax) \ %dx \]

2 possibilities \( \rightarrow \text{move a byte (b): bw} \)

\( \text{move 2 bytes (w): w} \)

\[ \text{mov mem\rightarrow mem not allowed: split in 2.} \]

\[ O_x100 \quad O_x101 \quad O_x102 \quad O_x103 \]

\[ 00 \quad AH \quad BB \quad 22 \]

\[ \text{movb mem-address, } \%al \]

\[ \text{movw mem-address, } \%ax \]

\[ \text{movl } -11- \quad , \quad \%eax \]

Stack Operations: Pg 3 02/17/16.

lea (Gerald Pg 6): Only take ec[1]

Arithmetic Operations: Gerald pg 7 O\text{\textregistered}4

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

\[ \text{value at } S = O_x1 \quad \text{value at } D = M[\%eax] = O_xFF \]

\[ \text{new value at } D = O_x1 + O_xFF = O_x100. \]

\[ O_x100 \quad O_x101 \quad O_x102 \quad O_x103 \]

\[ 00 \quad 00 \quad 00 \quad AB \]

\[ FF \quad 00 \quad 00 \quad 00 \]
Let's assume $0_{10}^{\text{h}} - 0_{10}^{\text{h}} 0_{10}^{\text{h}}$.

```
0x104 0x AB
\% ecx = 0x1
\% eax = 0x100.

add \% ecx, \% eax (Carry)
```

**Shift instruction**: $0 \leq k \leq 3$

**Left-Shift**: add zeros. $r_i$

3.4.8, 3.4.5 (Reading)

Control: Gerald Pg 8, 9, 10.

Worksheet.