Assembly I: Basic Operations

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Moving Data (1)

- **Moving data: movl source, dest**
  - Move 4-byte ("long") word
  - Lots of these in typical code

- **Operand types**
  - Immediate: constant integer data
    - Like C constant, but prefixed with ‘$’
    - e.g. $0x400, $-533
    - Encoded with 1, 2, or 4 bytes
  - Register: one of 8 integer registers
    - But %esp and %ebp reserved for special use
    - Others have special uses for particular instructions
  - Memory: 4 consecutive bytes of memory
    - Various “addressing modes”

\%
- eax
- ebx
- ecx
- edx
- esi
- edi
- esp
- ebp
Moving Data (2)

- **movl operand combinations**

  - Cannot do memory-memory transfers with single instruction

```
<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>C Analog</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>movl</strong>&lt;br&gt;Imm&lt;br&gt;Reg&lt;br&gt;Mem&lt;br&gt;Reg&lt;br&gt;Mem</td>
<td>movl $0x4,%eax&lt;br&gt;movl $-147,%eax&lt;br&gt;movl %eax,%edx&lt;br&gt;movl %eax,(%edx)&lt;br&gt;movl (%eax),%edx</td>
<td>temp = 0x4;&lt;br&gt;*p = -147;&lt;br&gt;temp2 = temp1;&lt;br&gt;*p = temp;&lt;br&gt;temp = *p;</td>
</tr>
</tbody>
</table>
```

C Analog examples:
- `temp = 0x4;`
- `*p = -147;`
- `temp2 = temp1;`
- `*p = temp;`
- `temp = *p;`
Simple Addressing Modes

- **Normal** \((R)\) \(\text{Mem}[\text{Reg}[R]]\)
  - Register R specifies memory address
  - e.g., \texttt{movl (%ecx), %eax}

- **Displacement** \(D(R)\) \(\text{Mem}[\text{Reg}[R]+D]\)
  - Register R specifies start of memory region
  - Constant displacement D specifies offset
  - e.g., \texttt{movl 8(%ebp), %edx}
Indexed Addressing Modes (1)

- **Most general form:**

  $$D(Rb, Ri, S) \rightarrow Mem[Reg[Rb]+S\times Reg[Ri]+D]$$

  - **D:** constant “displacement”: 1, 2, or 4 bytes
  - **Rb:** Base register: any of 8 integer registers
  - **Ri:** Index register: any, except for %esp & %ebp
  - **S:** Scale: 1, 2, 4, or 8

- **Special cases**

  - **(Rb,Ri)**  $$\rightarrow Mem[Reg[Rb]+Reg[Ri]]$$
  - **D(Rb,Ri)**  $$\rightarrow Mem[Reg[Rb]+Reg[Ri]+D]$$
  - **(Rb,Ri,S)**  $$\rightarrow Mem[Reg[Rb]+S\times Reg[Ri]]$$

  Useful to access arrays and structures
## Indexed Addressing Modes (2)

- Address computation example

<table>
<thead>
<tr>
<th>Expression</th>
<th>Computation</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0x8(%edx)$</td>
<td>$0xf000 + 0x8$</td>
<td>$0xf008$</td>
</tr>
<tr>
<td>($%edx,%ecx$)</td>
<td>$0xf000 + 0x100$</td>
<td>$0xf100$</td>
</tr>
<tr>
<td>($%edx,%ecx,4$)</td>
<td>$0xf000 + 4*0x100$</td>
<td>$0xf400$</td>
</tr>
<tr>
<td>$0x80(,%edx,2)$</td>
<td>$2*0xf000 + 0x80$</td>
<td>$0x1e080$</td>
</tr>
</tbody>
</table>
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}

swap:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx

    movl 12(%ebp),%ecx
    movl 8(%ebp),%edx
    movl (%ecx),%eax
    movl (%edx),%ebx
    movl %eax,(%edx)
    movl %ebx,(%ecx)

    movl -4(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
void swap(int *xp, int *yp) {
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}

Register Allocation (By compiler)

<table>
<thead>
<tr>
<th>Register</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ecx</td>
<td>yp</td>
</tr>
<tr>
<td>%edx</td>
<td>xp</td>
</tr>
<tr>
<td>%eax</td>
<td>t1</td>
</tr>
<tr>
<td>%ebx</td>
<td>t0</td>
</tr>
</tbody>
</table>

movl 12(%ebp),%ecx  # ecx = yp
movl 8(%ebp),%edx  # edx = xp
movl (%ecx),%eax  # eax = *yp (t1)
movl (%edx),%ebx  # ebx = *xp (t0)
movl %eax,(%edx) # *xp = eax
movl %ebx,(%ecx) # *yp = ebx
### Understanding Swap (2)

#### Register Allocation (By compiler)

<table>
<thead>
<tr>
<th>Register</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ecx</td>
<td>yp</td>
</tr>
<tr>
<td>%edx</td>
<td>xp</td>
</tr>
<tr>
<td>%eax</td>
<td>t1</td>
</tr>
<tr>
<td>%ebx</td>
<td>t0</td>
</tr>
</tbody>
</table>

#### Address Table

<table>
<thead>
<tr>
<th>Offset</th>
<th>Address</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0x124</td>
<td>%ecx</td>
</tr>
<tr>
<td>8</td>
<td>0x120</td>
<td>%edx</td>
</tr>
<tr>
<td>4</td>
<td>0x12c</td>
<td>%ecx</td>
</tr>
<tr>
<td>0</td>
<td>0x118</td>
<td>%ebx</td>
</tr>
<tr>
<td>-4</td>
<td>0x11c</td>
<td>%esi</td>
</tr>
<tr>
<td>0</td>
<td>0x114</td>
<td>%edi</td>
</tr>
<tr>
<td></td>
<td>0x110</td>
<td>%esp</td>
</tr>
<tr>
<td></td>
<td>0x10c</td>
<td>%ebp</td>
</tr>
<tr>
<td></td>
<td>0x108</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x104</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x100</td>
<td></td>
</tr>
</tbody>
</table>

- movl 12(%ebp),%ecx
  # ecx = yp
- movl 8(%ebp),%edx
  # edx = xp
- movl (%ecx),%eax
  # eax = *yp (t1)
- movl (%edx),%ebx
  # ebx = *xp (t0)
- movl %eax,(%edx)
- movl %ebx,(%ecx)
  # *xp = eax
  # *yp = ebx
### Understanding Swap (3)

#### Register Allocation (By compiler)

<table>
<thead>
<tr>
<th>Register</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ecx</td>
<td>yp</td>
</tr>
<tr>
<td>%edx</td>
<td>xp</td>
</tr>
<tr>
<td>%eax</td>
<td>t1</td>
</tr>
<tr>
<td>%ebx</td>
<td>t0</td>
</tr>
</tbody>
</table>

#### Code Snippets

```assembly
movl 12(%ebp),%ecx  # ecx = yp
movl 8(%ebp),%edx   # edx = xp
movl (%ecx),%eax    # eax = *yp (t1)
movl (%edx),%ebx    # ebx = *xp (t0)
movl %eax,(%edx)    # *xp = eax
movl %ebx,(%ecx)    # *yp = ebx
```
Understanding Swap (4)

```
movl 12(%ebp),%ecx  # ecx = yp
movl 8(%ebp),%edx  # edx = xp
movl (%ecx),%eax  # eax = *yp (t1)
movl (%edx),%ebx  # ebx = *xp (t0)
movl %eax,(%edx)  # *xp = eax
movl %ebx,(%ecx)  # *yp = ebx
```

<table>
<thead>
<tr>
<th>Offset</th>
<th>Address</th>
<th>Register Allocation (By compiler)</th>
</tr>
</thead>
<tbody>
<tr>
<td>yp</td>
<td>12</td>
<td>%eax 0x124</td>
</tr>
<tr>
<td>xp</td>
<td>8</td>
<td>%edx 0x120</td>
</tr>
<tr>
<td>%ebp</td>
<td>0</td>
<td>%ecx 0x11c</td>
</tr>
<tr>
<td></td>
<td>-4</td>
<td>%esi 0x118</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%edi 0x114</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%esp 0x110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>%ebp 0x104</td>
</tr>
</tbody>
</table>

```

Register | Variable
---------|--------
%ecx     | yp     
%edx     | xp     
%eax     | t1     
%ebx     | t0     

Address 0x124 0x120 0x11c 0x118 0x114 0x110 0x10c 0x108 0x104 0x100
Rtn adr  0x120 0x124
\[\text{movl} 12(\%ebp),\%ecx \quad \# \text{ecx} = \text{yp}\\ \text{movl} 8(\%ebp),\%edx \quad \# \text{edx} = \text{xp}\\ \text{movl} (\%ecx),\%eax \quad \# \text{eax} = *\text{yp} (t1)\\ \text{movl} (\%edx),\%ebx \quad \# \text{ebx} = *\text{xp} (t0)\\ \text{movl} \%eax,(\%edx) \quad \# *\text{xp} = \text{eax}\\ \text{movl} \%ebx,(\%ecx) \quad \# *\text{yp} = \text{ebx}\\
```
## Understanding Swap (5)

### Register Allocation (By compiler)

<table>
<thead>
<tr>
<th>Register</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ecx</td>
<td>yp</td>
</tr>
<tr>
<td>%edx</td>
<td>xp</td>
</tr>
<tr>
<td>%eax</td>
<td>t1</td>
</tr>
<tr>
<td>%ebx</td>
<td>t0</td>
</tr>
</tbody>
</table>

### Address

<table>
<thead>
<tr>
<th>Offset</th>
<th>yp</th>
<th>xp</th>
<th>%ebp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
<td>8</td>
<td>-4</td>
</tr>
<tr>
<td>12</td>
<td>0x120</td>
<td>0x124</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0x118</td>
<td>0x114</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0x108</td>
<td>0x104</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0x104</td>
<td>0x100</td>
<td></td>
</tr>
</tbody>
</table>

### Code

```assembly
movl 12(%ebp),%ecx  # ecx = yp
movl 8(%ebp),%edx   # edx = xp
movl (%ecx),%eax    # eax = *yp (t1)
movl (%edx),%ebx    # ebx = *xp (t0)
movl %eax,(%edx)    # *xp = eax
movl %ebx,(%ecx)    # *yp = ebx
```
Understanding Swap (6)

Register Allocation (By compiler)

<table>
<thead>
<tr>
<th>Register</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ecx</td>
<td>yp</td>
</tr>
<tr>
<td>%edx</td>
<td>xp</td>
</tr>
<tr>
<td>%eax</td>
<td>t1</td>
</tr>
<tr>
<td>%ebx</td>
<td>t0</td>
</tr>
</tbody>
</table>

Address

- %eax: 456
- %edx: 0x124
- %ecx: 0x120
- %ebx: 123
- %esi
- %edi
- %esp
- %ebp: 0x104

Offset

- yp: 12
- xp: 8
- %ebp: 0
- Rtn adr: -4

Instructions:

- movl 12(%ebp),%ecx  # ecx = yp
- movl 8(%ebp),%edx  # edx = xp
- movl (%ecx),%eax  # eax = *yp (t1)
- movl (%edx),%ebx  # ebx = *xp (t0)
- movl %eax,(%edx)  # *xp = eax
- movl %ebx,(%ecx)  # *yp = ebx
Understanding Swap (7)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Address</th>
<th>%eax</th>
<th>%edx</th>
<th>%ecx</th>
<th>%ebx</th>
<th>%esi</th>
<th>%edi</th>
<th>%esp</th>
<th>%ebp</th>
</tr>
</thead>
<tbody>
<tr>
<td>yp</td>
<td>0x120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xp</td>
<td>0x124</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td>0x100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x104</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x108</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x10c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x114</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x118</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x11C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x124</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Register Allocation (By compiler)

<table>
<thead>
<tr>
<th>Register</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ecx</td>
<td>yp</td>
</tr>
<tr>
<td>%edx</td>
<td>xp</td>
</tr>
<tr>
<td>%eax</td>
<td>t1</td>
</tr>
<tr>
<td>%ebx</td>
<td>t0</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{movl} & \ 12(\%ebp),\%ecx & \# \text{ ecx } = \text{ yp} \\
\text{movl} & \ 8(\%ebp),\%edx & \# \text{ edx } = \text{ xp} \\
\text{movl} & \ (%ecx),\%eax & \# \text{ eax } = *\text{ yp } (t1) \\
\text{movl} & \ (%edx),\%ebx & \# \text{ ebx } = *\text{ xp } (t0) \\
\text{movl} & \ %eax,(\%edx) & \# *\text{ xp } = \text{ eax} \\
\text{movl} & \ %ebx,(\%ecx) & \# *\text{ yp } = \text{ ebx} \\
\end{align*}
Understanding Swap (8)

Register Allocation (By compiler)

<table>
<thead>
<tr>
<th>Register</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ecx</td>
<td>yp</td>
</tr>
<tr>
<td>%edx</td>
<td>xp</td>
</tr>
<tr>
<td>%eax</td>
<td>t1</td>
</tr>
<tr>
<td>%ebx</td>
<td>t0</td>
</tr>
</tbody>
</table>

Address

<table>
<thead>
<tr>
<th>Address</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x124</td>
<td>%edx</td>
</tr>
<tr>
<td>0x120</td>
<td>%ecx</td>
</tr>
<tr>
<td>0x11c</td>
<td>%esi</td>
</tr>
<tr>
<td>0x118</td>
<td>%edi</td>
</tr>
<tr>
<td>0x114</td>
<td>%esp</td>
</tr>
<tr>
<td>0x110</td>
<td>%ebp</td>
</tr>
</tbody>
</table>

movl 12(%ebp),%ecx  # ecx = yp
movl 8(%ebp),%edx   # edx = xp
movl (%ecx),%eax    # eax = *yp (t1)
movl (%edx),%ebx    # ebx = *xp (t0)
movl %eax, (%edx)   # *xp = eax
movl %ebx, (%ecx)   # *yp = ebx

movl 456, %eax      # eax = 456
movl 0x124, %edx    # edx = 0x124
movl 0x120, %ecx    # ecx = 0x120
movl 123, %esi      # esi = 123
movl 123, %edi      # edi = 123
movl 0x104, %ebp    # ebp = 0x104
# Arithmetic/Logical Ops. (1)

## Two operands instructions

- **addl** $\text{Src, Dest}$  
  \[ \text{Dest} = \text{Dest} + \text{Src} \]

- **subl** $\text{Src, Dest}$  
  \[ \text{Dest} = \text{Dest} - \text{Src} \]

- **mull** $\text{Src, Dest}$  
  \[ \text{Dest} = \text{Dest} * \text{Src} \text{ (unsigned)} \]

- **imull** $\text{Src, Dest}$  
  \[ \text{Dest} = \text{Dest} * \text{Src} \text{ (signed)} \]

- **sall** $\text{Src, Dest}$  
  \[ \text{Dest} = \text{Dest} \ll \text{Src} (= \text{shll}) \]

- **sarl** $\text{Src, Dest}$  
  \[ \text{Dest} = \text{Dest} \gg \text{Src} \text{ (Arithmetic)} \]

- **shrl** $\text{Src, Dest}$  
  \[ \text{Dest} = \text{Dest} \gg \text{Src} \text{ (Logical)} \]

- **xorl** $\text{Src, Dest}$  
  \[ \text{Dest} = \text{Dest} \oplus \text{Src} \]

- **andl** $\text{Src, Dest}$  
  \[ \text{Dest} = \text{Dest} \& \text{Src} \]

- **orl** $\text{Src, Dest}$  
  \[ \text{Dest} = \text{Dest} \mid \text{Src} \]
Arithmetic/Logical Ops. (2)

- **One operand instructions**
  - incl Dest \( \text{Dest} = \text{Dest} + 1 \)
  -decl Dest \( \text{Dest} = \text{Dest} - 1 \)
  - negl Dest \( \text{Dest} = -\text{Dest} \)
  - notl Dest \( \text{Dest} = \sim\text{Dest} \)
Address Computation

- **leal** *Src, Dest*
  - *Src* is address mode expression
  - Set *Dest* to address denoted by expression

- **Uses**
  - Computing address without doing memory reference
    - e.g., translation of \( p = \&x[i]; \)
  - Computing arithmetic expressions of the form \( x + k*y \)
    - \( k = 1, 2, 4, \) or 8
Example: arith (1)

```c
int arith
(int x, int y, int z)
{
    int t1 = x+y;
    int t2 = z+t1;
    int t3 = x+4;
    int t4 = y * 48;
    int t5 = t3 + t4;
    int rval = t2 * t5;
    return rval;
}
```

```assembly
arith:
    pushl %ebp
    movl %esp,%ebp

    movl 8(%ebp),%eax
    movl 12(%ebp),%edx
    leal (%edx,%eax),%ecx
    leal (%edx,%edx,2),%edx
    sall $4,%edx
    addl 16(%ebp),%ecx
    imull %ecx,%eax

    movl %ebp,%esp
    popl %ebp
    ret
```
Example: arith (2)

```c
int arith
(int x, int y, int z)
{
    int t1 = x+y;
    int t2 = z+t1;
    int t3 = x+4;
    int t4 = y * 48;
    int t5 = t3 + t4;
    int rval = t2 * t5;
    return rval;
}
```

<table>
<thead>
<tr>
<th>Offset</th>
<th>Rtn adr</th>
<th>%ebp</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Old %ebp</td>
<td>%ebp</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>

```
movl 8(%ebp),%eax  # eax = x
movl 12(%ebp),%edx  # edx = y
leal (%edx,%eax),%ecx  # ecx = x+y (t1)
leal (%edx,%edx,2),%edx  # edx = 3*y
sall $4,%edx  # edx = 48*y (t4)
addl 16(%ebp),%ecx  # ecx = z+t1 (t2)
leal 4(%edx,%eax),%eax  # eax = 4+t4+x (t5)
imull %ecx,%eax  # eax = t5*t2 (rval)
```
Example: logical

```c
int logical(int x, int y)
{
    int t1 = x^y;
    int t2 = t1 >> 17;
    int mask = (1<<13) - 7;
    int rval = t2 & mask;
    return rval;
}

2^{13} = 8192, 2^{13} - 7 = 8185
```

**logical:**

```assembly
pushl %ebp
movl %esp,%ebp

movl 8(%ebp),%eax ; eax = x
xorl 12(%ebp),%eax ; eax = x^y  (t1)
sarl $17,%eax ; eax = t1>>17 (t2)
andl $8185,%eax ; eax = t2 & 8185

movl %ebp,%esp
popl %ebp
ret
```

Set Up

Body

Finish
CISC Properties

- CISC (Complex Instruction Set Computer)
  - Instruction can reference different operand types
    - Immediate, register, memory
  - Arithmetic operations can read/write memory
  - Memory reference can involve complex computation
    - $R_b + S*R_i + D$
    - Useful for arithmetic expressions, too.
  - Instructions can have varying lengths
    - IA-32 instructions can range from 1 to 15 bytes
Summary (1)

- Machine level programming
  - Assembly code is textual form of binary object code
  - Low-level representation of program
    - Explicit manipulation of registers
    - Simple and explicit instructions
    - Minimal concept of data types
    - Many C control constructs must be implemented with multiple instructions
Summary (2)

Machine Models

Data
1) char
2) int, float
3) double
4) struct, array
5) pointer

Control
1) loops
2) conditionals
3) switch
4) Proc. call
5) Proc. return

Machine Models

C

Data
1) char
2) int, float
3) double
4) struct, array
5) pointer

Control
1) loops
2) conditionals
3) switch
4) Proc. call
5) Proc. return

C

1) byte
2) 2-byte word
3) 4-byte long word
4) contiguous byte allocation
5) address of initial byte

1) branch/jump
2) call
3) ret

Assembly

Compiler