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"
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>Imm</td>
<td>Reg movl $0x4,%eax</td>
<td>temp = 0x4;</td>
</tr>
<tr>
<td>Mem</td>
<td>Reg movl $-147,(%eax)</td>
<td>*p = -147;</td>
</tr>
<tr>
<td>Reg</td>
<td>Reg movl %eax,%edx</td>
<td>temp2 = temp1;</td>
</tr>
<tr>
<td>Mem</td>
<td>Reg movl %eax,(%edx)</td>
<td>*p = temp;</td>
</tr>
<tr>
<td>Mem</td>
<td>Reg movl (%eax),%edx</td>
<td>temp = *p;</td>
</tr>
</tbody>
</table>
Simple Addressing Modes

- **Normal** \((R)\) \(\text{Mem}[\text{Reg}[R]]\)
  - Register R specifies memory address
  - e.g., `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., `movl 8(\%ebp), \%edx`
Indexed Addressing Modes (1)

**Most general form:**

\[ D(Rb, Ri, S) \rightarrow \text{Mem}[\text{Reg}[Rb] + S\cdot\text{Reg}[Ri] + D] \]

- **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 \text{Mem}[\text{Reg}[Rb] + \text{Reg}[Ri]]
- \(D(Rb, Ri)\) \rightarrow \text{Mem}[\text{Reg}[Rb] + \text{Reg}[Ri] + D]
- \((Rb, Ri, S)\) \rightarrow \text{Mem}[\text{Reg}[Rb] + S\cdot\text{Reg}[Ri]]

Useful to access arrays and structures
### 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>
Swap Example

```c
void swap(int *xp, int *yp) {
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

swap:

```assembly
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
```

Body

Setup

Finish
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>
<thead>
<tr>
<th>Address</th>
<th>%eax</th>
<th>%edx</th>
<th>%ecx</th>
<th>%esi</th>
<th>%edi</th>
<th>%esp</th>
<th>%ebp</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x124</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x104</td>
</tr>
<tr>
<td>0x120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x11c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x118</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x114</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x10c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x108</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x104</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Offset

<table>
<thead>
<tr>
<th>Offset</th>
<th>yp</th>
<th>xp</th>
<th>%ebp</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-4</td>
<td></td>
<td></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)  # *xp = eax
movl %ebx, (%ecx)  # *yp = ebx
Understanding Swap (3)

```
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)

#### Register Allocation (By compiler)

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

#### Code Snippet

```
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 (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>Address</th>
<th>%eax</th>
<th>%edx</th>
<th>%ecx</th>
<th>%ebx</th>
</tr>
</thead>
<tbody>
<tr>
<td>yp</td>
<td>12</td>
<td>0x124</td>
<td>0x120</td>
<td>0x11c</td>
<td></td>
</tr>
<tr>
<td>xp</td>
<td>8</td>
<td>0x124</td>
<td>0x118</td>
<td>0x114</td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td>0</td>
<td>0x104</td>
<td>0x108</td>
<td>0x10c</td>
<td>0x104</td>
</tr>
<tr>
<td></td>
<td>-4</td>
<td>0x100</td>
<td></td>
<td></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)   # *xp = eax
movl %ebx,(%ecx)   # *yp = ebx

Register

<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>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>yp</td>
<td>12</td>
</tr>
<tr>
<td>xp</td>
<td>8</td>
</tr>
<tr>
<td>%ebp</td>
<td>0</td>
</tr>
<tr>
<td>%ebp</td>
<td>-4</td>
</tr>
</tbody>
</table>
Understanding Swap (6)

```plaintext
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)

```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 (8)

Register Allocation (By compiler)

<table>
<thead>
<tr>
<th>Register</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td>t1</td>
</tr>
<tr>
<td>%edx</td>
<td>xp</td>
</tr>
<tr>
<td>%ecx</td>
<td>yp</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
Arithmetic/Logical Ops. (1)

- **Two operands instructions**
  - `addl` Src, Dest \[\text{Dest} = \text{Dest} + \text{Src}\]
  - `subl` Src, Dest \[\text{Dest} = \text{Dest} - \text{Src}\]
  - `mull` Src, Dest \[\text{Dest} = \text{Dest} \times \text{Src} \text{ (unsigned)}\]
  - `imull` Src, Dest \[\text{Dest} = \text{Dest} \times \text{Src} \text{ (signed)}\]
  - `sall` Src, Dest \[\text{Dest} = \text{Dest} \ll \text{Src} (= \text{shll})\]
  - `sarl` Src, Dest \[\text{Dest} = \text{Dest} \gg \text{Src} \text{ (Arithmetic)}\]
  - `shrl` Src, Dest \[\text{Dest} = \text{Dest} \ggg \text{Src} \text{ (Logical)}\]
  - `xorl` Src, Dest \[\text{Dest} = \text{Dest} \oplus \text{Src}\]
  - `andl` Src, Dest \[\text{Dest} = \text{Dest} \& \text{Src}\]
  - `orl` 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 $x + k*y$
    - $k = 1, 2, 4, \text{ or } 8$
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;
}
```

```
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
    leal 4(%edx,%eax),%eax
    imull %ecx,%eax
    movl %ebp,%esp
    popl %ebp
    ret
```

Set Up
Body
Finish
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;
}
```

Stack

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

Assembly Code:

- `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:
pushl %ebp
movl %esp,%ebp

movl 8(%ebp),%eax
xorl 12(%ebp),%eax
sarl $17,%eax
andl $8185,%eax

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

Set Up

Body

Finish

- movl 8(%ebp),%eax: \( \text{eax} = \text{x} \)
- xorl 12(%ebp),%eax: \( \text{eax} = \text{x}^\text{y} \) \( (t1) \)
- sarl $17,%eax: \( \text{eax} = \text{t1} \gg 17 \) \( (t2) \)
- andl $8185,%eax: \( \text{eax} = \text{t2} \& \text{8185} \)
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
    - \( R_b + S \times R_i + D \)
    - 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

Control

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

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

Machine Models

Data

Control

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