Class Announcements

1. Take backups of your C files periodically. Saves lot of work in case bad things happen.

2. Brief session on C Programming aspects relevant to Assignment 1 in later part of next lecture. (Turns out I cannot go into details because that is part of the assignment).

Lecture Overview

- Stack related Data Movement operations
- Data Movement example
- Arithmetic instructions
Increasing address

Stack "top" Stack "bottom"

Stack "top" Stack "bottom"

Stack "top" Stack "bottom"

pushl and popl

- pushl %ebp is equivalent to:
  subl $4, %esp
  movl %ebp, (%esp)

- popl %eax is equivalent to:
  movl (%esp), %eax
  addl $4, %esp

Data Movement Example
(Trace through during lecture)

```
.data
value:
    .long 52713
heapvar:
    .long 0x5000
.text
globl main
main:
    movl $103, %eax
    movl %eax, %esi
    movl value, %ebx
    Continued from left column:
    movl %esp, %ecx
    movl %eax, (%ecx)
    movl heapvar, %eax
    movl 8(%eax), %edx
    push %edx
    push $207
    pop %edi
    movl $3, %ecx
    movl (%eax, %ecx, 4), %edx
    ret
```

Stack Example: pushl, popl

```
Initially

pushl %eax

popl %edx
```

Arithmetic Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>leal S, D</td>
<td>(load effective address) D gets the address defined by S</td>
</tr>
<tr>
<td>inc D</td>
<td>D gets D + 1 (two's complement)</td>
</tr>
<tr>
<td>dec D</td>
<td>D gets D - 1 (two's complement)</td>
</tr>
<tr>
<td>neg D</td>
<td>D gets -D (two's complement additive inverse)</td>
</tr>
<tr>
<td>add S, D</td>
<td>D gets D + S (two's complement)</td>
</tr>
<tr>
<td>sub S, D</td>
<td>D gets D - S (two's complement)</td>
</tr>
<tr>
<td>imul S, D</td>
<td>D gets D * S (two's complement integer multiplication)</td>
</tr>
</tbody>
</table>
**More Arithmetic Instructions, with 64 bits of results**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>imul</code> $s$</td>
<td>$%edx$/$%eax$ gets 64-bit two's complement product of $%eax$ * $%eax$</td>
</tr>
<tr>
<td><code>mull</code> $s$</td>
<td>$%edx$/$%eax$ gets 64-bit unsigned product of $%eax$ * $%eax$</td>
</tr>
<tr>
<td><code>idiv</code> $s$</td>
<td>Two's complement division of $%edx$/$%eax$; $%edx$ gets remainder, and $%eax$ gets quotient</td>
</tr>
<tr>
<td><code>div</code> $s$</td>
<td>Unsigned division of $%edx$/$%eax$; $%edx$ gets remainder, and $%eax$ gets quotient</td>
</tr>
</tbody>
</table>

Notice implied use of $%eax$ and $%edx$.

---

**Examples**

Assume $%eax$ is $x$ and $%ecx$ is $y$ and $%edx=10$, address 10 has value 100

1. `leal 6(%eax), %edx :: ?`
2. `leal 9(%eax,%ecx,2), %edx :: ?`
3. `addl %ecx, (%edx) :: ?`
4. `decl %ecx :: ?`

---

**leal is commonly used to calculate addresses. Examples:**

- `leal 8(%eax), %edx`
  - 8 + contents of eax goes into edx
  - used for pointer arithmetic in C
  - very convenient for acquiring the address of an array element

- `leal (%eax, %ecx, 4), %edx`
  - contents of eax + 4 * contents of ecx goes into edx
  - even more convenient for addresses of array elements, where eax has base address, ecx has the index, and each element is 4 bytes

---

**Examples**

Assume $%eax$ is $x$ and $%ecx$ is $y$ and $%edx=10$, address 10 has value 100

1. `leal 6(%eax), %edx :: 6+x`
2. `leal 9(%eax,%ecx,2), %edx :: 9 + x + 2y`
3. `addl %ecx, (%edx) :: (y +100) stored @ address 10`
4. `decl %ecx :: (y-1) stored in %ecx`
Examples

Assume x at \%ebp+8, y at \%ebp+12, z at \%ebp+16

1 movl 16(%ebp), %eax  
   \text{z}

2 leal (%eax, %eax, 2), %eax  
   z*3

3 sall $4, %eax  
   t2 = z*48

4 movl 12(%ebp), %edx  
   y

5 addl 8(%ebp), %edx  
   t1 = x+y

6 andl $65535, %edx  
   t3 = t1 \& 0xFFFF

7 imull %edx, %eax  
   t4 = t2*t3