Midterm Exam Results

- Class A (Average = 150/230)
- Class B (Average = 137/230)
Notice

- **Midterm exam results**
  - Check out your score in GLS
  - Your exam papers and answer sheets are available in #400629.
  - If you have any claims, please visit my office during 5/10 ~ 5/14.

- **First programming assignment results**
  - They will be posted on the course homepage soon.
  - Any claims to 김형준 조교 (hjkim@cs1.skku.edu, #400629)
Notice

- **Project assignment #2**
  - Due date is 5/9 11:59PM
  - Send your code and document to cse2003skku@cs.l.skku.edu and cse2003skku@gmail.com
  - Any questions to 김형준 조교 (hjkim@cs.l.skku.edu)

- **Guest lectures**
  - Class on 5/3 by Prof. Hwansoo Han
  - Class on 5/6 by Prof. Joonwon Lee
Assembly III: Procedures

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
IA-32 Stack (1)

- **Characteristics**
  - Region of memory managed with stack discipline
  - Grows toward lower addresses
  - Register `%esp` indicates lowest stack address
    - address of top element
IA-32 Stack (2)

- **Pushing**
  - `pushl Src`
  - Fetch operand at `Src`
  - Decrement `%esp` by 4
  - Write operand at address given by `%esp`
IA-32 Stack (3)

- Popping
  - `popl Dest`
  - Read operand at address given by `%esp`
  - Increment `%esp` by 4
  - Write to `Dest`
**IA-32 Stack (4)**

- **Stack operation examples**

```plaintext
pushl %eax

%eax  213
%edx  555
%esp  0x108

popl %edx

%eax  213
%edx  555
%esp  0x104
```

```
%esp    %eax    %edx
0x108   0x10c   213
0x10c   0x108   123
0x110   0x10c   0x108

%esp    %eax    %edx
0x104   0x108   213
0x108   0x104   123
0x110   0x104   0x108
```
Procedure Control Flow

- Use stack to support procedure call and return

- Procedure call
  - call label
    - Push return address on stack
    - Jump to label
  - Return address value
    - Address of instruction beyond call

- Procedure return
  - ret
    - Pop address from stack
    - Jump to address
Procedure Call Example

804854e: e8 3d 06 00 00 call 8048b90 <main>
8048553: 50 pushl %eax

0x8048553 +0x0000063d = 0x8048b90

%esp 0x108
%esp 0x104
%eip 0x804854e
%eip 0x8048b90

%eip is program counter

call 8048b90

8048553: 50 pushl %eax

%eip is program counter
Procedure Return Example

\[
\begin{array}{c}
8048591: \text{ c3} \\
\text{ret}
\end{array}
\]

%esp: 0x104
%eip: 0x8048591

%esp: 0x104
%eip: 0x8048553

%eip is program counter
Stack-based Languages

- **Languages that support recursion**
  - e.g., C, Pascal, Java, etc.
  - Code must be “Reentrant”
    - Multiple simultaneous instantiations of single procedure
  - Need some place to store state of each instantiation
    - Arguments, local variables, return pointer

- **Stack discipline**
  - State for given procedure needed for limited time
    - From when called to when return
  - Callee returns before caller does

- **Stack allocated in frames**
  - State for single procedure instantiation
Stack Frames (1)

Code Structure

```cpp
yoo(...) {
    ...
    who();
    ...
}

who(...) {
    ...
    amI();
    ...
    amI();
}

amI(...) {
    ...
    ...
    amI();
    ...
}
```

- Procedure `amI` recursive

Call Chain

```
yoo
    who
        amI
            amI
```

CSE2003: System Programming | Spring 2010 | Jin-Soo Kim (jinsookim@skku.edu)
Stack Frames (2)

- **Contents**
  - Return information
  - Arguments
  - Local variables & temp space

- **Management**
  - Space allocated when enter procedure
    - “set-up” code
  - Deallocated when return
    - “finish” code

- **Pointers**
  - Stack pointer `%esp` indicates stack top
  - Frame pointer `%ebp` indicates start of current frame
Stack Frames (3)

Call Chain

```
yoo(...) {
  .
  .
  who();
  .
  .
}
```
Stack Frames (4)

```c
void who(...) {
    amI();
    amI();
}
```

Call Chain

Frame Pointer

%ebp

Stack Pointer

%esp

who

yoo

CSE2003: System Programming | Spring 2010 | Jin-Soo Kim (jinsookim@skku.edu)
Stack Frames (5)

Call Chain

```
stack_frame1
  amI(...)
  {
    •
    •
    •
    •
  }
```

```
stack_frame2
  yoo
  who
  amI
```

```
stack_frame3
  Frame Pointer
  %ebp
```

```
stack_frame4
  Stack Pointer
  %esp
```

```
stack_frame5
  •
  •
```

CSE2003: System Programming | Spring 2010 | Jin-Soo Kim (jinsookim@skku.edu)
Stack Frames (6)

Call Chain

```c
amI(...) {
  .
  .
  amI();
  .
  .
}
```

Frame Pointer %ebp

Stack Pointer %esp
Stack Frames (7)

Call Chain

```
void amI (...) {
  ...;
  amI();
  ...;
}
```

Frame Pointer

Stack Pointer

%ebp

%esp

yoo

who

amI

amI

amI

amI
Stack Frames (8)

```
amI(...) 
{
    .
    .
    amI();
    .
    .
}
```

Call Chain

```
yoo
  who
  amI
  amI
  amI
  Frame Pointer %ebp
  Stack Pointer %esp
```
Stack Frames (9)

Call Chain

```
amI(...) {
    
    amI();
    
}
```

Frame Pointer %ebp

Stack Pointer %esp
Stack Frames (10)

Call Chain

```c
who(...) {
    .
    ami();
    .
}
```

Frame Pointer
%ebp

Stack Pointer
%esp

CSE2003: System Programming | Spring 2010 | Jin-Soo Kim (jinsookim@skku.edu)
Stack Frames (11)

```
amI(...) {
    .
    .
    .
}
```

Call Chain

```
Frame Pointer %ebp
Stack Pointer %esp
```

CSE2003: System Programming | Spring 2010 | Jin-Soo Kim (jinsookim@skku.edu)
Stack Frames (12)

```c
void who(...) {
    amI();
    amI();
}
```

Call Chain:

- Frame Pointer %ebp
- Stack Pointer %esp

Diagram showing the call chain and stack frames.
Stack Frames (13)

```
yoo(...) 
{ 
  .  
  .  
  who();  
  .  
} 
```

Call Chain

- Frame Pointer `%ebp`
- Stack Pointer `%esp`

```
yoo
  who
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
  ami
 ami
IA-32/Linux Stack Frame

- **Current stack frame ("Top" to Bottom)**
  - Parameters for function about to call
    - "Argument build"
  - Local variables
    - If can’t keep in registers
  - Saved register context
  - Old frame pointer

- **Caller stack frame**
  - Return address
    - Pushed by call instruction
  - Arguments for this call
Revisiting swap (1)

int zip1 = 15213;
int zip2 = 91125;

void call_swap()
{
    swap(&zip1, &zip2);
}

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

Calling swap from call_swap

call_swap:
    • • •
    pushl $zip2    # Global Var
    pushl $zip1    # Global Var
    call swap
    • • •

Resulting Stack

%esp

Rtn adr

&zip2
&zip1

CSE2003: System Programming  |  Spring 2010  |  Jin-Soo Kim (jinsookim@skku.edu)
Revisiting swap (2)

```c
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
```
Swap Setup (1)

**Entering Stack**

- `%ebp`
- ...
- `%esp`
- `&zip2`
- `&zip1`
- `Rtn adr`

**Resulting Stack**

- `%ebp`
- ...
- `%esp`
- `YP`
- `xp`
- `Rtn adr`
- `Old %ebp`

**swap:**

```
pushl %ebp
movl %esp,%ebp
pushl %ebx
```
Swap Setup (2)

Entering Stack

Resulting Stack

\[
\text{swap:} \\
\text{pushl} \ %\text{ebp} \\
\text{movl} \ %\text{esp},%\text{ebp} \\
\text{pushl} \ %\text{ebx}
\]
Swap Setup (3)

Entering Stack

\[ %ebp \]
\[ \text{\&zip2} \]
\[ \text{\&zip1} \]
\[ \text{Rtn adr} \]

Resulting Stack

\[ %ebp \]
\[ \text{YP} \]
\[ \text{xp} \]
\[ \text{Rtn adr} \]
\[ \text{Old %ebp} \]
\[ \text{Old %ebx} \]

\[ \text{swap:} \]
\[ \text{pushl %ebp} \]
\[ \text{movl %esp,%ebp} \]
\[ \text{pushl %ebx} \]
Effect of swap Setup

Entering Stack

Resulting Stack

Body

movl 12(%ebp),%ecx    # get yp
movl 8(%ebp),%edx     # get xp
...
swap Finish (1)

- Observation
  - Saved & restored register %ebx

```plaintext
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
```
swap Finish (2)

swap’s Stack

Offset
12
8
4
0
-4

YP
xp
Rtn adr
Old %ebp
Old %ebx
%ebp
%esp

movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
swap Finish (3)

swap’s Stack

Offset

12             yp
8              xp
4              Rtn adr
0             Old %ebp
-4           Old %ebx

movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
## swap Finish (4)

### swap’s Stack

<table>
<thead>
<tr>
<th>Offset</th>
<th>YP</th>
<th>Rtn adr</th>
<th>Old %ebp</th>
<th>Old %ebx</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Exiting Stack

<table>
<thead>
<tr>
<th>Offset</th>
<th>YP</th>
<th>Rtn adr</th>
<th>Old %ebp</th>
<th>Old %ebx</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Observation
- Saved & restored register %ebx
- Didn’t do so for %eax, %ecx, or %edx

```assembly
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
```
Register Saving Conventions (1)

- When procedure yoo() calls who():
  - yoo is the **caller**, who is the **callee**

- Can register be used for temporary storage?

  ```
  yoo:
  ... 
  movl $15213, %edx
  call who
  addl %edx, %eax
  ... 
  ret
  ```

  ```
  who:
  ... 
  movl 8(%ebp), %edx
  addl $91125, %edx
  ... 
  ret
  ```

  - Contents of register **%edx** overwritten by **who**
Register Saving Conventions (2)

- Conventions
  - “Caller save”
    - Caller saves temporary in its frame before calling
  - “Callee save”
    - Callee saves temporary in its frame before using
- Integer registers
  - Two have special uses:
    - %ebp, %esp
  - Three managed as callee-save:
    - %ebx, %esi, %edi
    - Old values saved on stack prior to using
  - Three managed as caller-save:
    - %eax, %edx, %ecx
    - Do what you please, but expect any callee to do so, as well
  - Register %eax also stores returned value
Recursive Factorial: rfact

- Registers
  - `%eax` used without first saving
  - `%ebx` used, but save at beginning & restore at end

```c
int rfact(int x)
{
    int rval;
    if (x <= 1)
        return 1;
    rval = rfact(x-1);
    return rval * x;
}
```

```
rfact:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
    movl 8(%ebp),%ebx
    cmpl $1,%ebx
    jle .L78
    leal -1(%ebx),%eax
    pushl %eax
    call rfact
    imull %ebx,%eax
    jmp .L79
    .align 4
    .L78:
        movl $1,%eax
    .L79:
        movl -4(%ebp),%ebx
        movl %ebp,%esp
        popl %ebp
        ret
```
rfact Stack Setup

Entering Stack

rfact:

pushl %ebp
movl %esp,%ebp
pushl %ebx

41
CSE2003: System Programming | Spring 2010 | Jin-Soo Kim (jinsookim@skku.edu)
rfact Body

- Registers
  - \%ebx: stored value of \( x \)
  - \%eax
    - Temporary value of \( x-1 \)
    - Returned value from \( \text{rfact}(x-1) \)
    - Returned value from this call

Recursion

```c
int rfact(int x)
{
    int rval;
    if (x <= 1)
        return 1;
    rval = rfact(x-1);
    return rval * x;
}
```

```assembly
movl 8(%ebp),%ebx       # ebx = x
cmp1 $1,%ebx           # Compare x : 1
jle .L78
lea -1(%ebx),%eax      # eax = x-1
pushl %eax             # Push x-1
call rfact             # rfact(x-1)
imull %ebx,%eax         # rval * x
jmp .L79
.L78:
movl $1,%eax           # return val = 1
.L79:
```

CSE2003: System Programming | Spring 2010 | Jin-Soo Kim (jinsookim@skku.edu)
rfact Recursion

```
leal -1(%ebx),%eax
```

```
x
Rtn adr
Old %ebp
Old %ebx

%eax
x-1
%ebx
x

pushl %eax

call rfact
```

```
x
Rtn adr
Old %ebp
Old %ebx

%eax
x-1
%ebx
x

%esp

Old %ebp

%esp

%esp

%esp
rfact Result

Return from Call

```
  x
Rtn adr
Old %ebp %ebp
Old %ebx %esp
x-1
```

```
imull %ebx,%eax
```

```
  x
Rtn adr
Old %ebp %ebp
Old %ebx %esp
x-1
```

```
%eax (x-1)!
%ebx x
```

```
%eax x!
%ebx x
```

Assume that rfact(x-1) returns (x-1)! in register %eax
Summary

- **The stack makes recursion work**
  - Private storage for each instance of procedure call
    - Instantiations don’t clobber each other
    - Addressing of locals + arguments can be relative to stack positions
  - Can be managed by stack discipline
    - Procedures return in inverse order of calls

- **Procedures = Instructions + Conventions**
  - Call / Ret instructions
  - Register usage conventions
    - Caller / Callee save
      - %ebp and %esp
  - Stack frame organization conventions