Class Announcements

1. How was Midterm1? Easy, Hard?

2. Any suggestions for Midterm2?
Lecture Overview

1. Intro to Functions and Stacks
2. Instructions used for for Function Calls
What we need to know how to do. . .
(what the compiler must be able to implement)

1. call
2. return
3. AR and local variables
4. return value
5. parameters
Important Note: In the following slides for this lecture the stack is represented as growing upwards with lower addresses at the top and higher addresses at the bottom. This is the opposite of what we have seen and will see in this course.

double words are pushed and popped

dedicated register %esp contains address of item currently at top of stack (TOS)
pushl  *

does   %esp <- %esp - 4
mvsl   *, ( %esp)

popl  *

does   movl  ( %esp), *
        %esp <- %esp + 4

THE STACK
1. call

- remember the return address
- go to fcn

This is such a common operation that the x86 architecture supports it with a single instruction:

```
call fcn
```

does the equivalent of

```
push %eip
jmp fcn
```

```
addr 0
%esp
```
2. return

use the return address pushed onto the stack

`ret`

does the equivalent of

`popl %eip`
3. incorporate AR

For example, assume we need AR space for 3 ints. 

`gcc` on x86 allocates AR space in multiples of 16 bytes.

Before fcn starts, but after the call instruction

After fcn prologue
prologue code

```assembly
pushl %ebp
movl %esp, %ebp
subl $16, %esp
```

Before fcn starts, but after the call instruction

After fcn prologue
epilogue code
leave
does movl %ebp, %esp
    popl %ebp
ret
does popl %eip
Put local variables into AR:

```c
void b() {
    int x, y, z;
    x = 1;
    y = 2;
    z = 3;
    c();
}
```

```assembly
b:  pushl %ebp   prologue
    movl %esp, %ebp
    subl $16, %esp
    movl $1, -12(%ebp)
    movl $2, -8(%ebp)
    movl $3, -4(%ebp)
    call c
    leave      epilogue
    ret
```

before epilogue

```
prev %ebp
ra
```
4. **return value**

On x86, return value goes in %eax (by convention)

```c
int b() {
    b:

    c();
    return 4;
}

call c
movl $4, %eax
leave
ret
```
5. parameters

No room in registers on the x86, so parameters go onto the stack.
Caller allocates space and places copies (for call by value). Child retrieves and uses copies.

```c
main () {
    a( 1, 2, 3);
}
main:  pushl %ebp
       movl %esp, %ebp
       subl $12, %esp
       movl $1, (%esp)
       movl $2, 4(%esp)
       movl $3, 8(%esp)
       call a
       leave
       ret
```
Current frame

Caller’s frame

Saved registers, local variables, and temporaries

Argument build area

Return address

Argument 1

Argument n

Stack “bottom”

Earlier frames

Stack “top”

Frame pointer

Increasing address

Caller’s frame

Current frame

Stack pointer

Saved %ebp

Argument

+4

+4n

+8

%ebp

%esp