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 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)

### 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  PC
   jmp fcn
   ```

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.

prologue code
pushl %ebp
movl %esp, %ebp
subl $16, %esp

epilogue code
leave
does movl %ebp, %esp
popl %ebp
ret
does popl %eip

Put local variables into AR:

void b() {
    b: pushl %ebp prologue
    int x, y, z;
    movl %esp, %ebp
    x = 1;
    subl $16, %esp
    y = 2;
    movl $1, -12(%ebp)
    z = 3;
    movl $2, -8(%ebp)
    c();
    movl $3, -4(%ebp)
    call c
    leave epilogue
    ret
}
4. **return value**
On x86, return value goes in $eax (by convention)

```c
int b() {
    b:

c();    call c
return 4;  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.

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