CS 354 - Machine Organization & Programming
Tuesday Nov 7th, and Thursday Nov 9th,2023

Midterm Exam - Thurs Nov 9th, 7:30 - 9:30 pm

- **UW ID and #2 required, room information sent via email (bring copy to exam)**
- **closed book, no notes, no electronic devices (e.g., calculators, phones, watches) see "Midterm Exam 2" on course site Assignments for topics**

A10 e2_cheatsheet.pdf

Homework hw4: DUE on or before Monday, Nov 6

p4AQuestions: DUE on or before Monday, Nov 6

Homework hw5: will be DUE on or before Monday, Nov 13

Project p4B: DUE on or before Friday, Nov 10

Project p5: DUE on or before Friday Apr 22

Learning Objectives

- identify and describe conventions for IA-32 registers and cond codes ZF, SF, OF, CF
- trace and describe how conditional assembly instructions and execution
- trace and describe how repetition is achieved in ASM and Mach Code
- trace and describe how control is transferred to a function call
- trace and describe how control is returned from a function call

This Week

Finish L18 Outline (Instructions) CMP and TEST, Condition Codes SET, Jumps, Encoding Targets, Converting Loops	The Stack from a Programmer's Perspective The Stack and Stack Frames Instructions - Transferring Control Register Usage Conventions Function Call-Return Example
Next Week: Finish Stack Frames B&O 3.7 Intro - 3.7.5 3.8 Array Allocation and Access 3.9 Heterogeneous Data Structures	

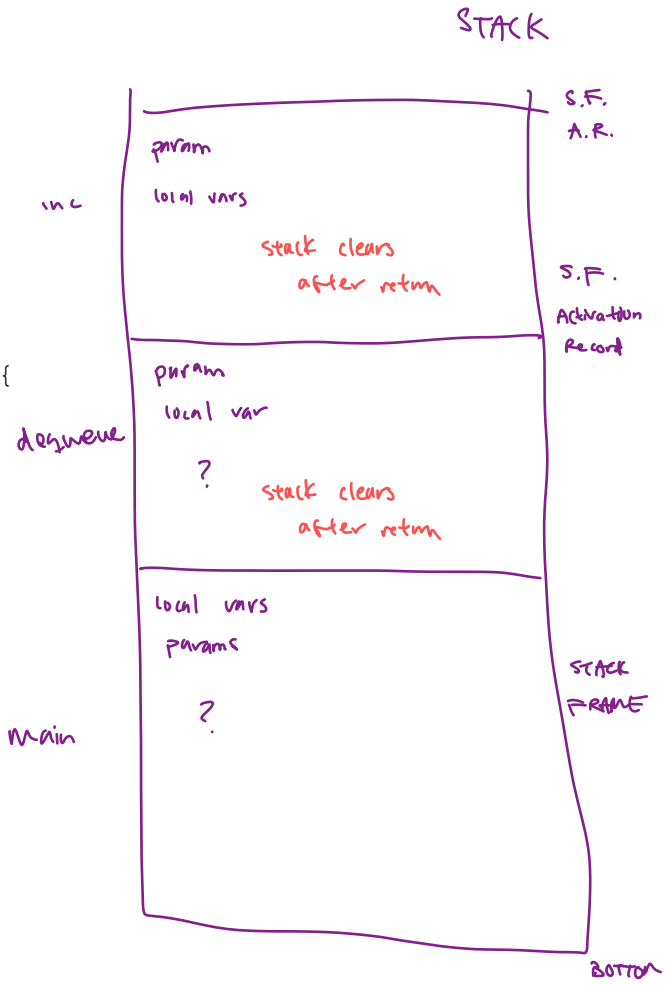
The Stack from a Programmer's Perspective

Consider the following code:

```
int inc(int index, int size) {
   int incindex = index + 1;
   if (incindex == size) return 0;
   return incindex;
}

int dequeue(int *queue, int *front,
        int rear, int *numitems, int size) {
   if (*numitem == 0) return -1;
   int dqitem = queue[*front];
   *front = inc(*front, size);
   *numitems -= 1;
   return dqitem;
}

int main(void) {
   int queue[5] = {11,22,33};
   int front = 0;
   int rear = 2;
   int numitems = 3;
   int qitem = dequeue(queue, &front, rear,
        &numitems, 5);
   ...
```



What does the compiler need to do to make function calls work?

- transfer control to callee & remember the ret addr
- handle passing args
- allocate & free stack frames
- " " " params & local vars
- handle return value
- other stuff

The Stack and Stack Frames

Stack Frame Activation Record

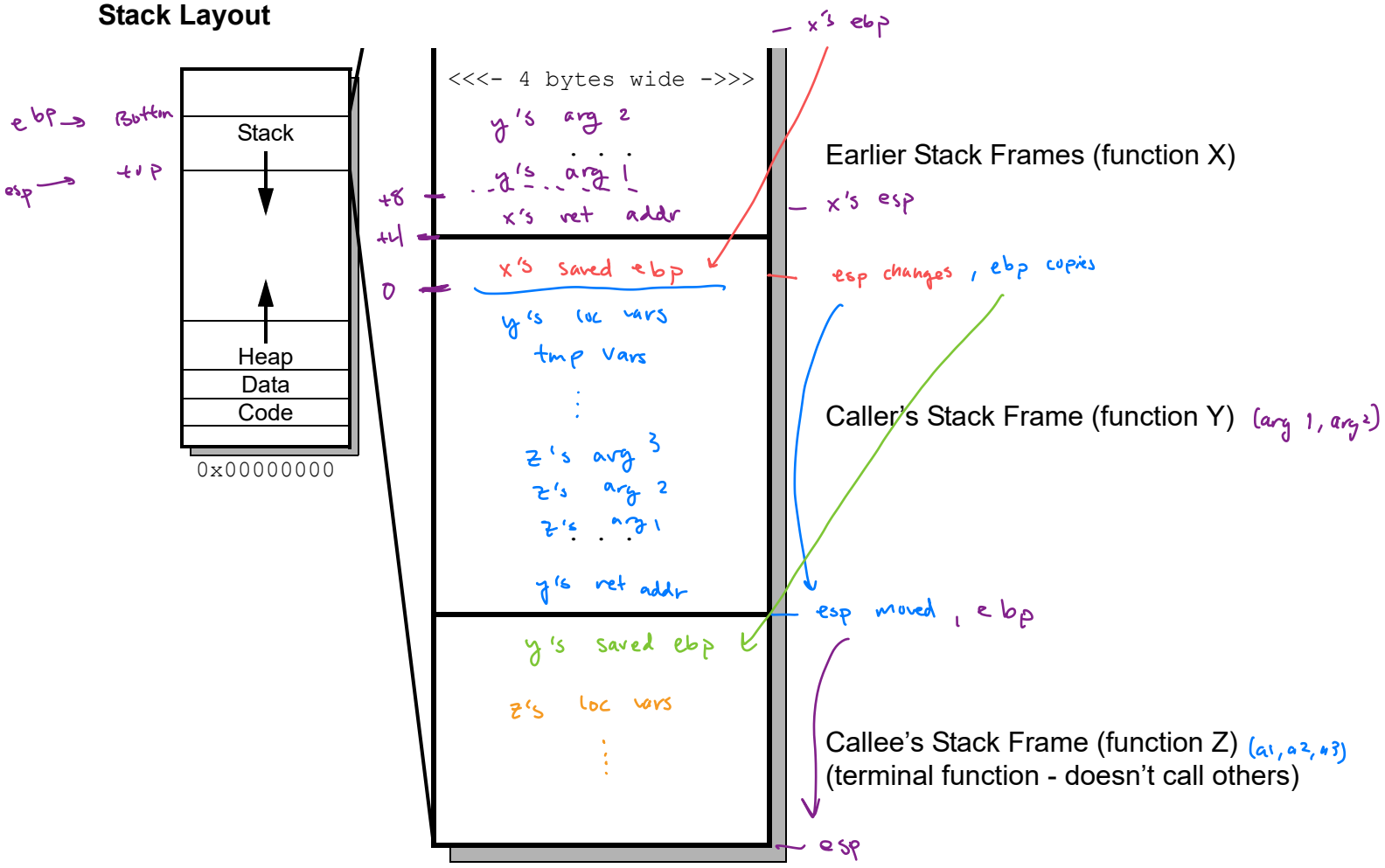
a block of stack mem used by a func call

IA-32: multiple of 16 byte

%ebp base ptr - ptr to base = bottom " of cur. S.F.

%esp stack ptr - pts to "top" of stack

Stack Layout



❋ *A Callee's args* are in caller's stack frame

→ What is the offset from the %ebp to get to a callee's first argument? 8 (%ebp)

→ When are local variables allocated on the stack? (and not in registers)

1. not enough register

2. arrays, structs, unions - complex struct

3. addr. of operator, so data has an addr

Instructions - Transferring Control

Flow Control

function call: like unconditional jumps

```
call *Operand
```

indir call

```
call Label
```

dir call

steps (for both forms of call)

1. push ret addr to stack ≡ pushl %eip
2. jmp to callee func ≡ jmp * operand (or label)

function return:

```
ret
```

step

1. jmp to ret addr that is popped off the stack

① popl %eip

Stack Frames

allocate stack frame: no special cmd

subl +$X, %esp # where X is size of S.F.

free stack frame:

```
leave
```

free's callee's S.F

steps

1. Remove all of callee's S.F. except for caller ebp

   movl %ebp, %esp

2. Restore caller's S.F.

   popl %ebp

Register Usage Conventions

Return Value

%eax accumulator

Frame Base Pointer `%ebp`

callee uses to access callee's args 8(%ebp) 1st arg

access local vars -4(%ebp)

Stack Pointer `%esp`

caller uses to set up args for func call

save the return address

callee uses to restore the ret addr

save & restore caller's %ebp

Registers and Local Variables

→ Why use registers? FAST!, but # of size is limited to 1, 2, 4 bytes

→ Potential problem with multiple functions using registers? CONFLICTS

- caller & callee must be consistent approach to prevent overwriting

IA-32

caller-save: %eax, %ecx, %edx - caller saves before

callee-save: %ebx, %esi, %edi - callee saves & restore

Function Call-Return Example

1. update eip
2. execute instr. (update mem and reg)

```
int dequeue(int *queue, int *front, int rear, int *numitems, int size) {
   if (*numitem == 0) return -1;
   int dqitem = queue[*front];
   *front = inc(*front, size);

   *numitems -= 1;
   return dqitem;
}
int inc(int index, int size) {


   int incindex = index + 1;
   if (incindex == size) return 0;
   return incindex;
}
```

index (1st arg), size (2nd arg)

%eax

1ab setup calleE's args
2 call the calleE function
a save caller's return address
b transfer control to calleE
7 caller resumes, assigns return value

3 allocate callee's stack frame
a save calleR's frame base
b set callee's frame base
c set callee's top of stack
4 callee executes ...

5 free callee's stack frame
a restore calleR's top of stack
b restore calleR's frame base
6 transfer control back to calleR

CALL code in `dequeue`

addr, instr

```
1a 0x0_2C  movl index,(%esp)
 b 0x0_2E  movl size,4(%esp)
2  0x0_30  call inc
 a  pushl %eip   (save ret addr on "top" stack)
 b  jmp 0x_F0    (to func)   = mov 0xF0 to %eip
```

RETURN code in `dequeue`

```
7  0x0_55 movl %eax,(%ebx)
```

CALL code in `inc`

```
3a 0x0_F0  pushl %ebp
 b 0x0_F2  movl %esp,%ebp
 c 0x0_F4  subl $12,%esp
4  0x0_F6  execute inc function's body
```

RETURN code in `inc`

```
5  0x0_FA  leave   // free callee's SF
 a    movl %ebp, %esp
 b    popl %ebp   <- copy from stack, update esp
6  0x0_FB  ret
        popl %eip
```

Function Call-Return Example

Execution Trace of Stack and Registers