CS 354 - Machine Organization & Programming
Tuesday April 4th, and Thursday April 6th,2023

W10 Activity: e2 cheatsheet

Midterm Exam - Thurs April 6th, 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**

Homework hw4: DUE on or before Monday, Apr 3

Homework hw5: will be DUE on or before Monday, Apr 10

Project p4A: DUE on or before Friday, Mar 31

Project p4B: DUE on or before Friday, Apr 7

Project p5: DUE on or before Friday Apr 22

Last Week

C, Assembly, & Machine Code Low-level View of Data Registers Operand Specifiers & Practice L18-7 Instructions - MOV, PUSH, POP	Operand/Instruction Caveats Instruction - LEAL Instructions - Arithmetic and Shift ---------- END of Exam 2 Material ---------- Instructions - CMP and TEST, Condition Codes

This Week: From L18: Instructions - 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: Stack Frames B&O 3.7 Intro - 3.7.5 3.8 Array Allocation and Access 3.9 Heterogeneous Data Structures	

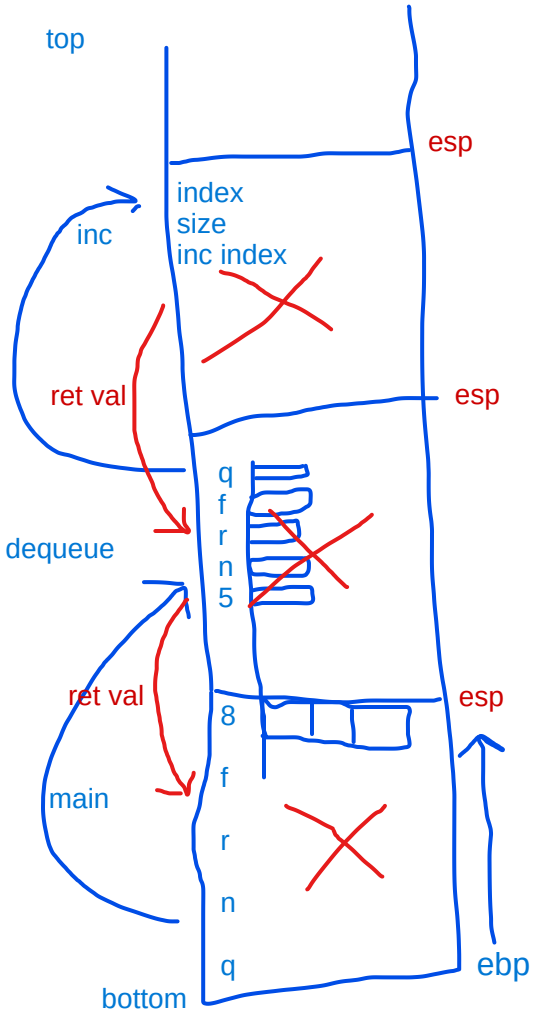
The Stack from a Programmer's Perspective

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
- handle passing arguments
- alloc/free stack frame
- alloc/free parameters and locals
- handle return value
- "other details"

The Stack and Stack Frames

Stack Frame a.k.a. Activation Record

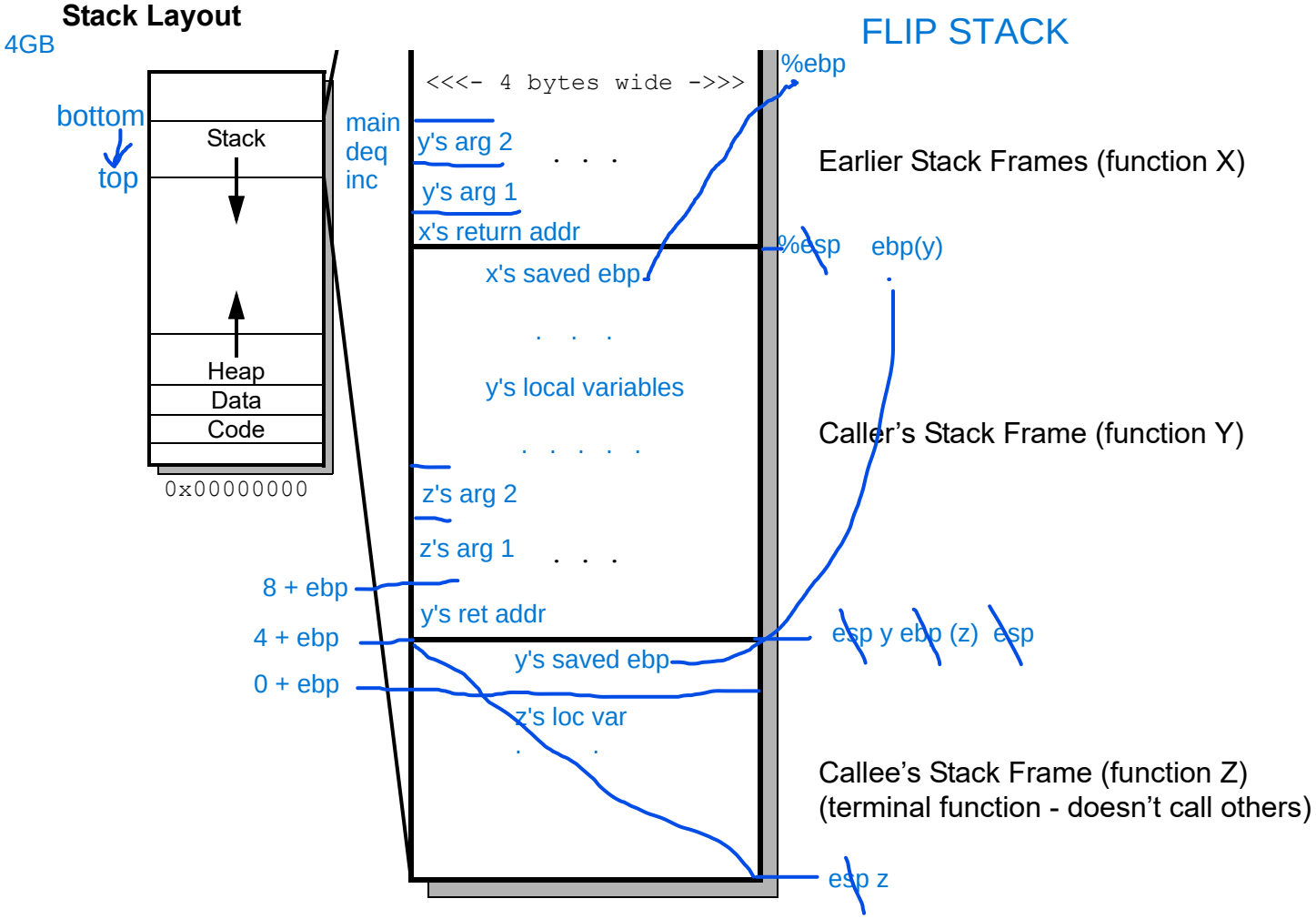
A block of memory used by a single function call

IA-32: must be multiple of 16

%ebp Base pointer register - points to bottom of stack frame

%esp Stack Pointer - points to top of stack frame

Stack Layout



- *A Callee's args* are in the 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?

1. not enough registers
2. are arrays, structs, or other complex data
3. code uses addressof &, so data has an address

Instructions - Transferring Control

Flow Control

function call:

```
call *Operand
```
indirect call

```
call Label
```
direct call

steps (for both forms of call)

1. push return address onto stack pushl %eip subl $4, %esp movl %eip, (%esp)
   ?^

2. jmp to start of called function
   jmp *operand
   jmp label

function return:

```
ret
```

step

1. jmp to ret addr popl %eip movl (%esp), %eip addl $4, %esp

Stack Frames

allocate stack frame:

No special instructions
use subl $_x_, %esp size in bytes of S.F.

free stack frame:

free's callee's Stack Frame

```
leave
```

steps

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

   ≡ movl %ebp, %esp

2. restore caller's S.F.

   popl %ebp

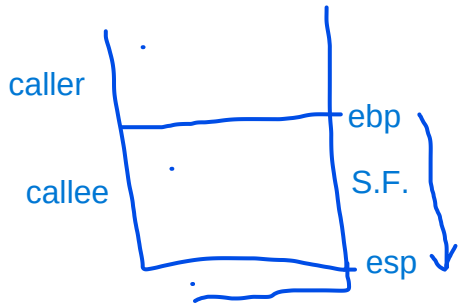
Register Usage Conventions

eax, edx, ecx, ebx
esi, edi, ebp, esp

Return Value

%eax

Frame Base Pointer `%ebp`



callee uses to %ebp to

1. to access callee's arguments
2. to access local variables

Stack Pointer `%esp`

caller uses to

1. set up args for function calls
2. save ret addr

callee uses to

1. restore ret addr
2. to save/restore caller's S.F.

Registers and Local Variables

→ Why use registers?

FAST! but data size is 1, 2, 4 bytes

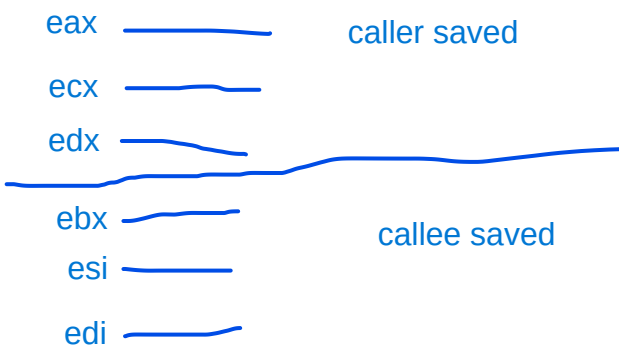
→ Potential problem with multiple functions using registers?

registers shared - conflict?
caller & callee must agree, consistent approach
save/restore register values

IA-32

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

callee-save: %ebx, %esi, %edi



Function Call-Return Example

```
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;
}
```

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`

1a `0x0_2C movl index,(%esp)`

b `0x0_2E movl size,4(%esp)`

2 `0x0_30 call inc`

a pushl %eip

b jmp 0x0_f0

subl $4, %esp
movl %eip, (%esp)

RETURN code in `dequeue`

ret addr

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 S.F., restore caller's S.F.

a movl %ebp, %esp

b popl %ebp

6 `0x0_FB ret`

popl %eip

Function Call-Return Example

Execution Trace of Stack and Registers