CS 354 - Machine Organization & Programming
Tuesday, November 12, 2019

Project p4b (~4%): DUE TOMORROW at 10 pm on Wednesday, November 13th

Last Time

The Stack from a Programmer's Perspective
The Stack and Stack Frames
Instructions - Transferring Control

Today

Register Usage Conventions
Function Call-Return Example
Recursion
Stack Allocated Arrays

Next Time

More Arrays, Structures, Pointers in Assembly
Read: B&O 3.8, 3.9
See pointers.pdf and recursion.pdf in Files section on course website

Register Usage Conventions

Return Value

Frame Base Pointer `%ebp`

callee uses to

Stack Pointer `%esp`

caller uses to

callee uses to

Registers and Local Variables

→ Why use registers?

→ Potential problem with multiple functions using registers?

IA-32

caller-save:

callee-save:

Function Call-Return Example

CALLER

```
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);
```

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

```
   *numitems -= 1;
   return dqitem;
}
```

CALLEE

```
int inc(int index, int size) {
```

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

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

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

CALLER

CALL code in `dequeue`

```
1a 0x_07C  movl index,(%esp)
 b 0x_07E  movl size,4(%esp)
2  0x_080  call inc
 a
 b
```

RETURN code in `dequeue`

```
7  0x_085 movl %eax,(%ebx)
```

CALLEE

CALL code in `inc`

```
3a 0x_110  pushl %ebp
 b 0x_112  movl %esp,%ebp
 c 0x_114  subl $12,%esp
```

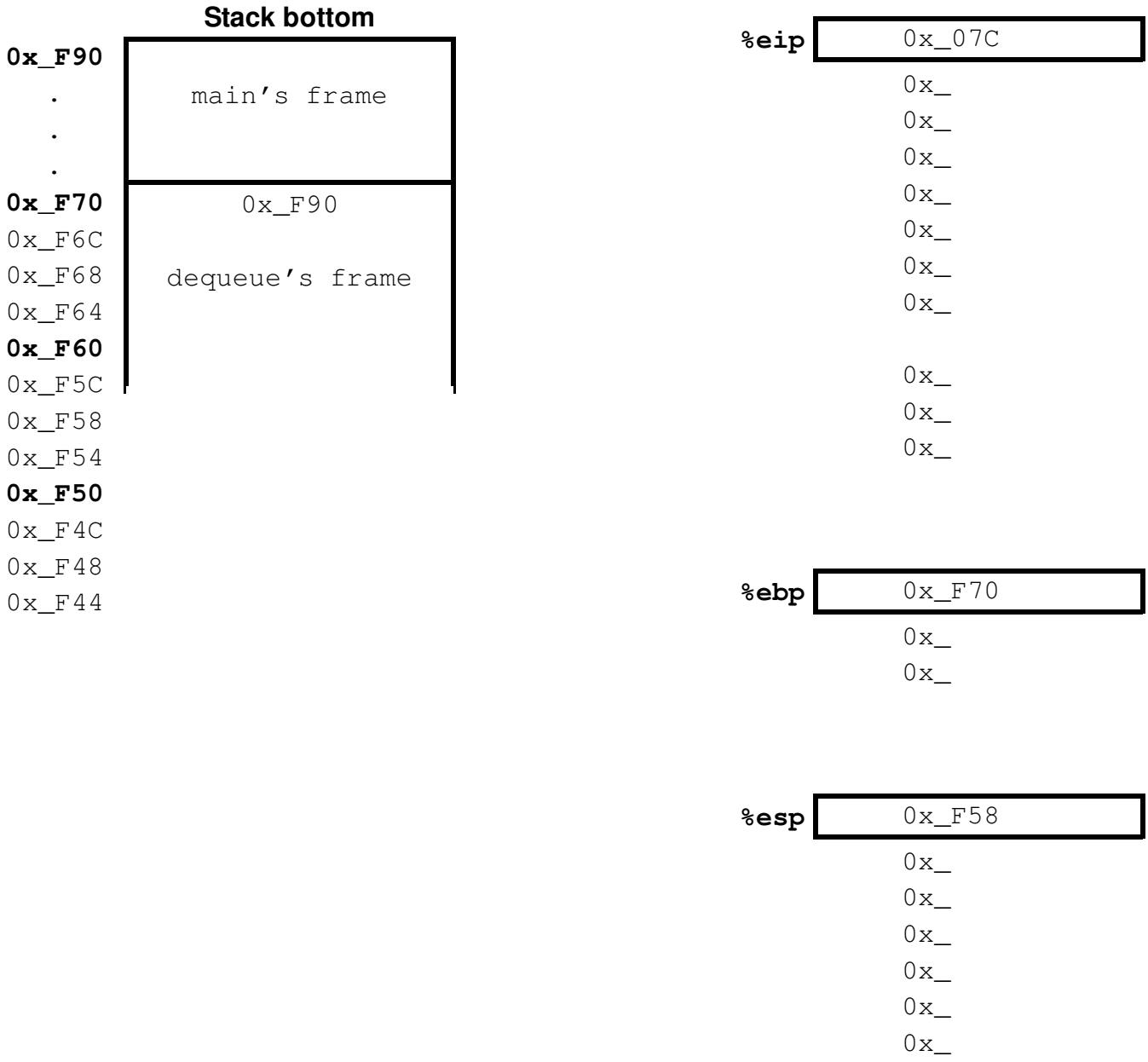
4 `0x_116` execute `inc` function's body

RETURN code in `inc`

```
5  0x_11A  leave
 a
 b
6  0x_11B  ret
```

Function Call-Return Example

Execution Trace of Stack and Registers



Recursion

Use a stack trace to determine the result of the call `fact(3)`:

```
int fact(int n) {
   int result;
   if (n <= 1) result = 1;
   else        result = n * fact(n - 1);
   return result;
}
```

direct recursion

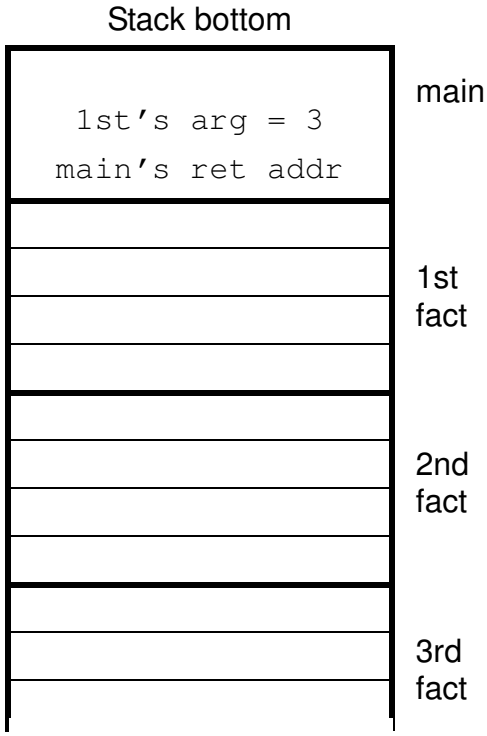
recursive case

base case

"infinite" recursion

Assembly Trace

```
fact:
   pushl %ebp
   movl %esp, %ebp
   pushl %ebx
   subl $4,%esp

   movl 8(%ebp),%ebx
   movl $1,%eax

   cmpl $1,%ebx
   jle .L1

   leal -1(%ebx),%eax
   movl %eax,(%esp)
   call fact

   imull %ebx,%eax

.L1:
   addl $4,%esp
   popl %ebx
   popl %ebp
   ret
```



- *"Infinite" recursion causes*

Stack Allocated Arrays in C

Recall Array Basics

T `A[`*N*`];` where *T* is the element datatype of size *L* bytes and *N* is the number of elements

1.

2.

❋ *The elements of* `A`

Recall Array Indexing and Address Arithmetic

```
&A[i]
```

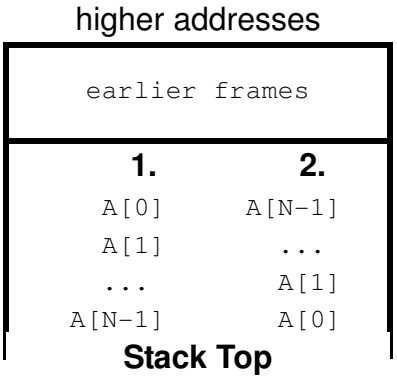
→ For each array declarations below, what is L (element size), the address arithmetic for the ith element, and the total size of the array?

	C code	L	address of ith element	total array size
1.	`int I[11]`			
2.	`char C[7]`			
3.	`double D[11]`			
4.	`short S[42]`			
5.	`char *C[13]`			
6.	`int **I[11]`			
7.	`double *D[7]`			

Stack Allocated Arrays in Assembly

Arrays on the Stack

→ How is an array laid out on the stack? Option 1 or 2:



❋ *The first element (index 0) of an array*

Accessing 1D Arrays in Assembly

Assume array's start address in %edx and index is in %ecx

```
movl (%edx, %ecx, 4), %eax
```

→ Assume `I` is an `int` array, `S` is a `short int` array, for both the array's start address is in %edx, and the index `i` is in %ecx. Determine the element type and instruction for each:

C code type assembly instruction to move C code's value into %eax

1. `I`
2. `I[0]`
3. `*I`
4. `I[i]`
5. `&I[2]`
6. `I+i-1`
7. `*(I+i-3)`
8. `S[3]`
9. `S+1`
10. `&S[i]`
11. `S[4*i+1]`
12. `S+i-5`