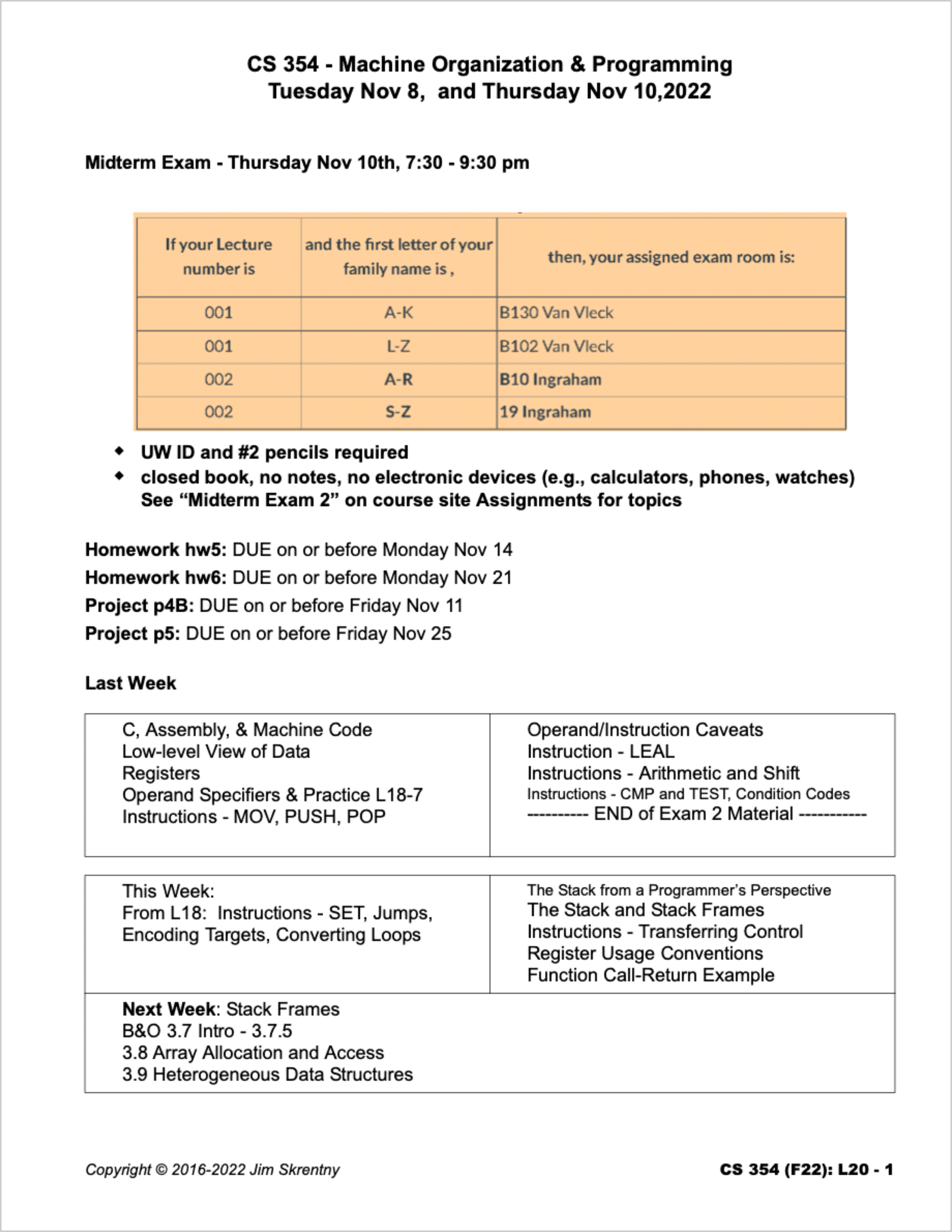
outlineL20-w10TR-student

Thursday, November 17, 2022 12:57 PM

outlineL20-w10TR-st...



CS 354 - Machine Organization & Programming
Tuesday Nov 8, and Thursday Nov 10,2022

Midterm Exam - Thursday Nov 10th, 7:30 - 9:30 pm

If your Lecture number is	and the first letter of your family name is ,	then, your assigned exam room is:
001	A-K	B130 Van Vleck
001	L-Z	B102 Van Vleck
002	**A-R**	**B10 Ingraham**
002	**S-Z**	**19 Ingraham**

- **UW ID and #2 pencils required**
- **closed book, no notes, no electronic devices (e.g., calculators, phones, watches)**
  See “Midterm Exam 2” on course site Assignments for topics

Homework hw5: DUE on or before Monday Nov 14

Homework hw6: DUE on or before Monday Nov 21

Project p4B: DUE on or before Friday Nov 11

Project p5: DUE on or before Friday Nov 25

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 Instructions - CMP and TEST, Condition Codes ---------- END of Exam 2 Material -----------

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	

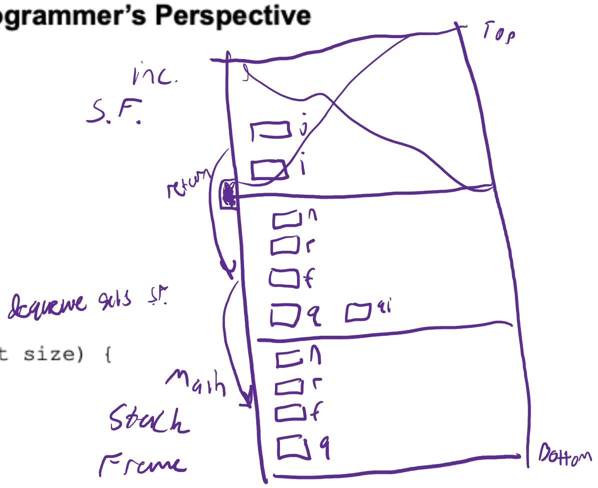
The Stack from a Programmer's Perspective

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 the calle
- has to pass arguments
- alloc and free stack frames
- alloc / free parameters and locals
- handle return value
- other details

The Stack and Stack Frames

Stack Frame aka Activation Record

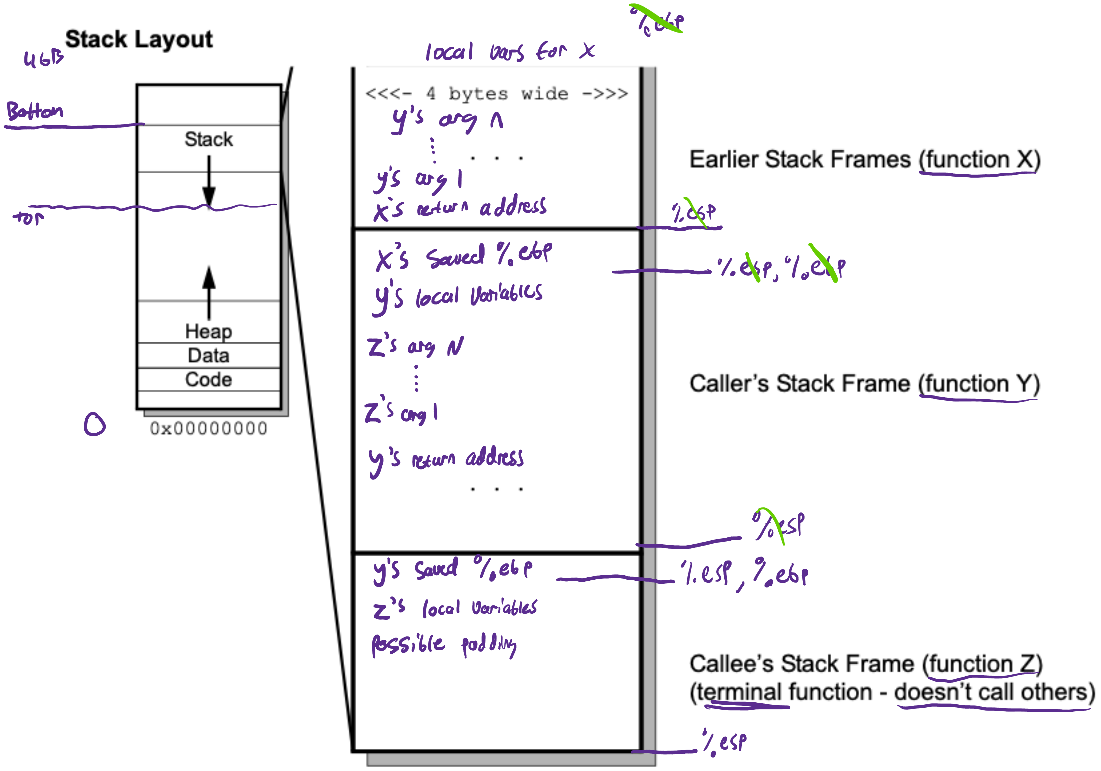
Block of memory used by a single func. call

IA-32: must be a multiple of 16 bytes

%ebp Base pointer register points to bottom 4 bytes of S.F.

%esp Stack Pointer register – points to top of stack

Stack Layout



❋ *A Callee's args* are in the caller's S.F.

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

Go back 2 words (2 words · 4 bytes = 8) so +8

→ When are local variables allocated on the stack?

1. not enough registers
2. are arrays, structs or other complex datatypes
3. Code uses address-of &, so data has mem address

Instructions - Transferring Control

Flow Control

function call:

`call *Operand` indirect call

`call Label` direct call

steps (for both forms of call)

1. push ret. addr onto stack

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

2. jump to start of callee function

   jmp *operand
   jmp label

function return:

`ret`

step

1. jump to return address that is popped off stack

   popl %eip [movl (%esp), %eip
   addl $4, %esp

Stack Frames

allocate stack frame:

No special instructions

Use: subl $X, %esp ← size in bytes of new stack frame

free stack frame:

`leave` free calle S.F.

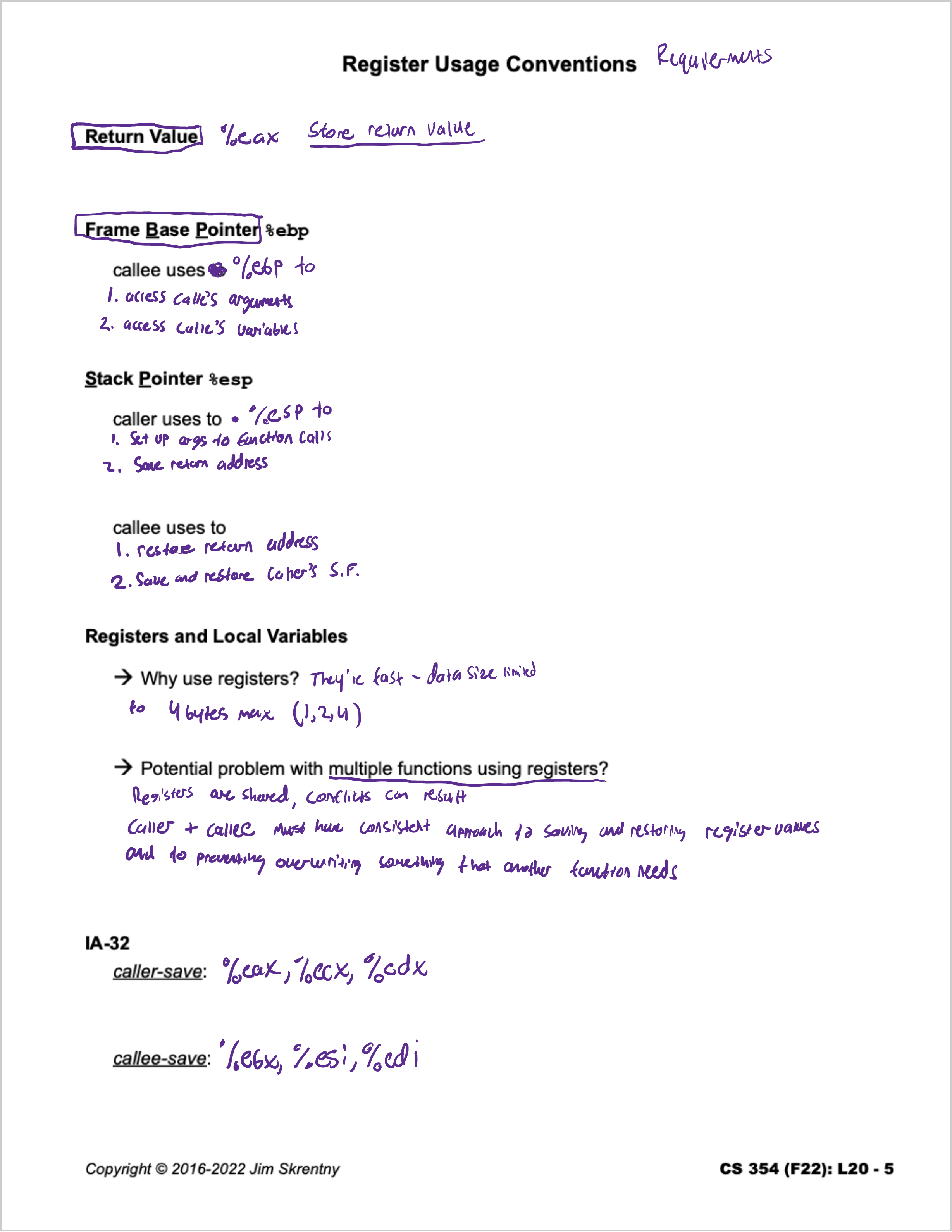
steps

1. remove all of calle's S.F. except caller's %ebp

   equiv: movl %ebp, %esp

2. restore the caller's frame

   popl %ebp



Register Usage Conventions Requirements

Return Value %eax Store return value

Frame Base Pointer `%ebp`

callee uses %ebp to
1. access calle's arguments
2. access calle's variables

Stack Pointer `%esp`

caller uses to • %esp to
1. Set up args to function calls
2. Save return address

callee uses to
1. restore return address
2. Save and restore caller's S.F.

Registers and Local Variables

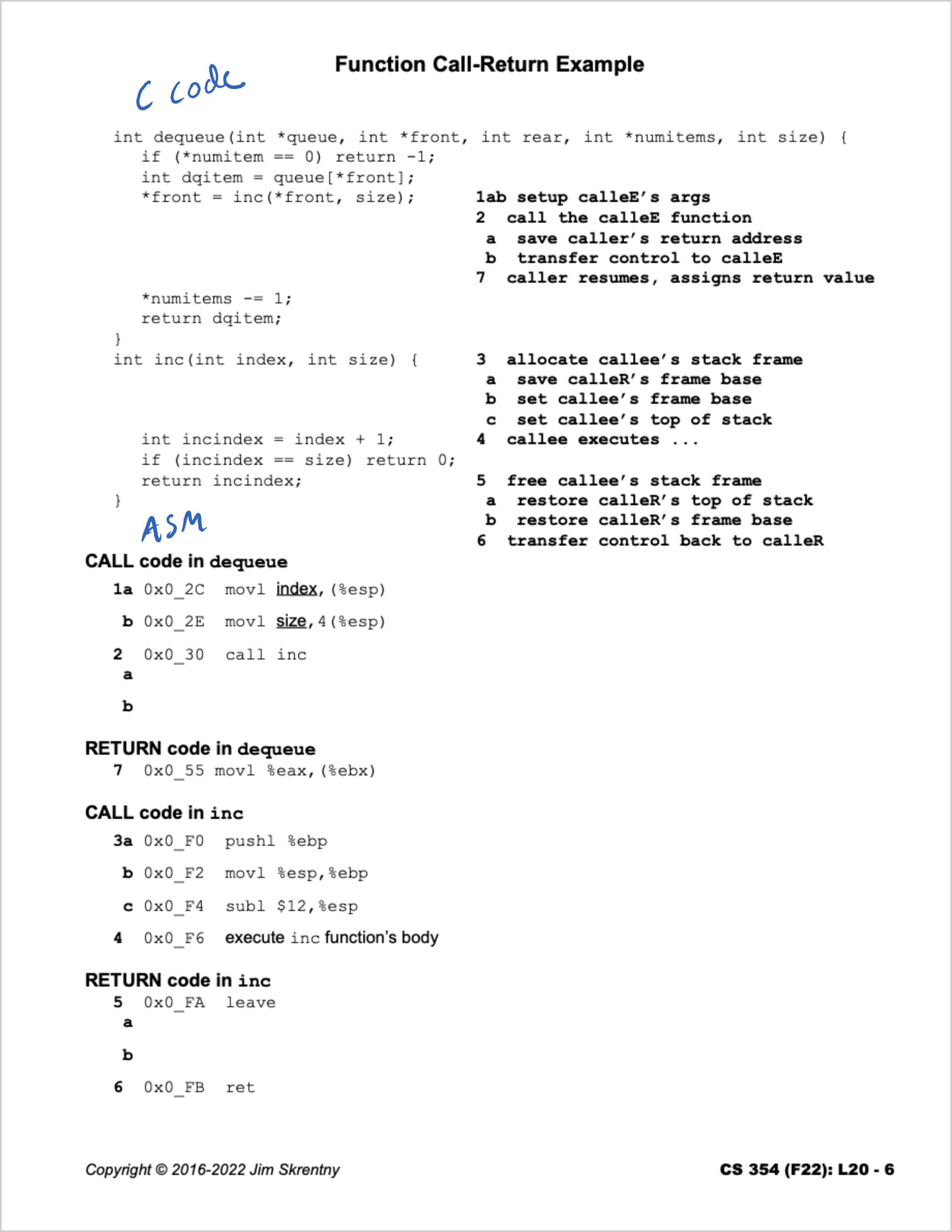
→ Why use registers? They're fast - data size limited to 4 bytes max (1,2,4)

→ Potential problem with multiple functions using registers?
Registers are shared, conflicts can result
Caller + callee must have consistent approach to saving and restoring register values and to preventing overwriting something that another function needs

IA-32

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

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



Function Call-Return Example

C code

```
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
```

ASM

CALL code in dequeue

```
1a 0x0_2C  movl index,(%esp)
 b 0x0_2E  movl size,4(%esp)
2  0x0_30  call inc
 a
 b
```

RETURN code in dequeue

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

CALL code in inc

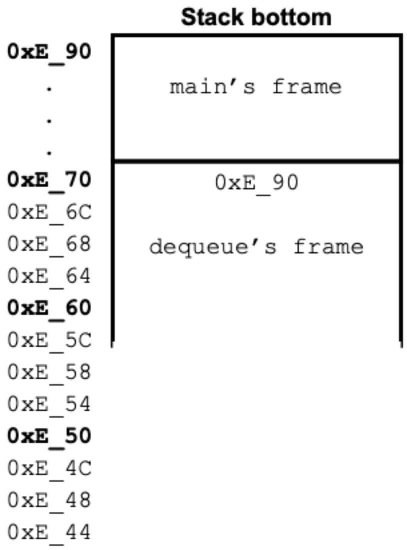
```
3a 0x0_F0  pushl %ebp
 b 0x0_F2  movl %esp,%ebp
 c 0x0_F4  subl $12,%esp
4  0x0_F6  execute inc function's body
```

RETURN code in inc

```
5  0x0_FA  leave
 a
 b
6  0x0_FB  ret
```

Function Call-Return Example

Execution Trace of Stack and Registers



Stack bottom

0xE_90
.
.
.
0xE_70
0xE_6C
0xE_68
0xE_64
0xE_60
0xE_5C
0xE_58
0xE_54
0xE_50
0xE_4C
0xE_48
0xE_44

main’s frame

0xE_90

dequeue’s frame

%eip 0x0_2C

0x0_
0x0_
0x0_
0x0_
0x0_
0x0_
0x0_

0x0_
0x0_
0x0_

%ebp 0xE_70

0xE_
0xE_

%esp 0xE_58

0xE_
0xE_
0xE_
0xE_
0xE_
0xE_