Adam Worthington
CS354
HW 0

1. What happens when Alice() calls Bob()?

- The three arguments passed in are pushed onto the stack from the rightmost argument to the leftmost. Each of these pushed increments the SP to the new top of the stack location.
- The return address is set to pushing an incremented IP onto the stack for bookkeeping. This means that once the stack has finished executing Bob() it will return to the caller's next line.
- The base pointer is now pushed onto the stack to keep a record of where Alice()'s stack frame is. The base pointer is then changed to the stack pointer. This is the start of the callee's stack frame.
- The stack now reserves space for uninitialized local variables (eg: "int x;"). After the BP is finished the stack needs to allocate space for the other local variables, and decrement the stack pointer.
- The AX register will be the register generally used to communicate between function for return values. These values can be either direct values of indirect references.
- After the function Bob() stores its result it needs to do some clean up. It will erase local variables by popping the stack and modifying the stack pointer. It will restore the base pointer with the value of the stack pointer from when we stored the caller's base pointer earlier. Finally it will jump to the return address stored on the stack to resume execution of the caller's function.
- A few extra mentions to tie up loose ends of the questions: The Callee knows how to reference the arguments because they are stored directly on the stack before the caller executes the jmp to the callee. Local variables are stored on the stack, however some variables such as allocated structs are stored on the "heap".

2.

```
 1        .file   "callbob.c"
```
Directive Used for assembler

```
 2        .text
```
Directive Used for assembler

```
 3        .globl  main
```
Directive Used for assembler

```
 4        .type   main, @function
```
Directive Used for assembler

```
 5 main:
```

How the program knows where to start, after initialization main will always be called.

6 pushl %ebp

Pushes the base pointer onto the stack, in our case the l is to signify it is 32 bit.

7 movl %esp, %ebp

Move the stack pointer into the current base pointer

8 subl $12, %esp

Creates room on the stack. the 12 comes from each integer requiring 4 bytes, so we need to make room for 3 integers of 12 bytes.

9 movl $5, 4(%esp)

Moves the constant 5 (furthest most passed in argument) 4 bytes above the stack pointer.

10 movl $2, (%esp)

Moves the constant 2 onto the stack pointer.

11 call bob

jmp to bob, pushes the IP for the return address.

12 movl %eax, -4(%ebp)

Moves the returned value from eax register 4 below the base pointer

13 leave

Cleans up the current stack frame. Resets SP to BP + 4 and then resets the BP to callee's BP

14 ret

Function finishes

15 .size main, .-main

Directive Used for assembler

16 .globl bob

Directive Used for assembler

17 .type bob, @function

Directive Used for assembler

18 bob:

Name of function for jmp to reference

19 pushl %ebp

Saves the main function's base pointer on the stack

20 movl %esp, %ebp
Moves the stack pointer to the base pointer to initialize a new stack frame for the bob function

21 subl $4, %esp
Creates stack for the local variable (int z) on the stack frame.

22 movl $3, -4(%ebp)
Puts the constant 3 in that allocated space

23 movl 12(%ebp), %eax
Move argument 2 into the ax register

24 addl %eax, -4(%ebp)
Add argument 2 (5) to the local variable 2 (3), and store in the space allocated for z

25 movl -4(%ebp), %eax
Move the value of z to the return register AX so the callee can reference it

26 leave
Cleans up the current stack frame. Resets SP to BP + 4 and then resets the BP to callee's BP

27 ret
Function finishes

28 .size bob, .-bob
Directive Used for assembler

29 .ident "GCC: (Gentoo 4.8.4 p1.5, pie-0.6.1) 4.8.4"
Directive Used for assembler

30 .section .note.GNU-stack,"",@progbits
Directive Used for assembler