Gray Houston
CS 354
Dr. Park

Homework 0

1.)

When alice invokes bob, there are a few values that are pushed onto the stack at the end of alice's stack frame. First, the 3 arguments that are passed to bob are pushed onto the stack in reverse order such that the last argument is located in the highest memory address of all the arguments and the first argument has the lowest memory address of the arguments. Finally, a pointer to the return address is pushed onto the stack. The next location in memory will be the location of the frame pointer (ebp) and the start of bob's stack frame.

In the caller's stack frame, %ebp is the pointer which refers to the start of the stack frame. Throughout the entire lifetime of this stack frame (function invocation), the value of the base pointer never changes. Therefore, it can be used as a reference point from which the function arguments can be found on the stack. By adding multiples of 4 to the frame pointer, you can get a reference to the arguments which are at the end of alice's stack frame. Additionally, by subtracting multiples of 4 from the base pointer, you can access local variables in the current stack frame.

In the caller, the stack pointer %esp refers to the boundary where memory on the stack is considered to be "allocated". If an address occurrs after the stack pointer, it is considered to be allocated on the stack, otherwise it is not. To allocate more space on the stack, the stack pointer can be subtracted by a multiple of 4, thereby adding more memory to what is considered to be in the stack frame. If bob had 2 local variables, the stack pointer would be decremented by 8. The first variable could be referenced by -4(%ebp) and the second local variable would -8(%ebp).

The convention for returning function values in x86 is to store the return value in the %eax register. Right before the callee function returns to the caller, the return value is moved into the %eax register. Once execution has returned to the caller function, it can retrieve the value of the %eax register to access the return value of the function it called.

2.)

```
        .file   "callbob.c"
        .def    ___main;        .scl    2;      .type   32;     .endef
        .text
        .globl  _main
        .def    _main;  .scl    2;      .type   32;     .endef
_main:
        pushl   %ebp            ; save the previous bae pointer on the stack
        movl    %esp, %ebp      ; set the base pointer to the current value of the
stack pointer
        subl    $12, %esp       ; subtract 12 from the stack pointer (allocating 3
variables)
        call    ___main         ; invoke the main function
        movl    $5, 4(%esp)     ; move the constant 5 into the 2nd function argument
        movl    $2, (%esp)      ; move the constant 2 into the 1st method argument
        call    _bob            ; invoke the bob function
```

```
        movl    %eax, -4(%ebp)  ; save bob's return value into the first local
variable on the sta\
ck
        leave                   ; restore callee's registers
        ret
        .globl  _bob
        .def    _bob;   .scl    2;      .type   32;     .endef
_bob:
        pushl   %ebp            ; save the prior base pointer onto the stack
        movl    %esp, %ebp      ; set the base pointer to point to the stack pointer
        subl    $4, %esp        ; allocate a local variable on the stack
        movl    $3, -4(%ebp)    ; set the first local variable to the value 3
        movl    12(%ebp), %eax  ; move the value of the 2nd argument into the %eax
register
        addl    %eax, -4(%ebp)  ; add the 2nd argument and local variable and store it
in the loca\
l variable
        movl    -4(%ebp), %eax  ; move the value of the local variable into %eax to
return to the \
callee
        leave                   ; restore callee's registers
        ret
        .ident  "GCC: (GNU) 4.9.3"
```