

Wednesday, October 5

Name _____

Email _____

6.035 Fall 2016

Miniquiz #13

5 minutes

The x86-64 Linux calling convention can be summarized as follows.

- The caller uses registers to pass the first 6 arguments to the callee. Given the arguments in left-to-right order, the order of registers used is: `%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8`, and `%r9`. Any remaining arguments are passed on the stack in reverse order so that they can be popped off the stack in order.
- The callee is responsible for perserving the value of registers `%rbp`, `%rbx`, and `%r12-r15`, as these registers are owned by the caller. The remaining registers (e.g., `%r8-r11`) are owned by the callee and are available for general use.
- The callee places its return value in `%rax` and is responsible for cleaning up its local variables as well as for removing the return address from the stack.

Consider the following x86-64 assembly code for a function `foo`.

```
foo:
    enter    $(8*2), $0
    mov     %rdi, -8(%rbp)

    mov     -8(%rbp), %r12
    add     $3, %r12
    mov     %r12, -16(%rbp)

    mov     -16(%rbp), %rax
    leave
    ret
```

Does `foo` adhere to the Linux calling convention? If not, what is wrong with the code? If necessary, rewrite the code such that it adheres to the Linux calling convention.