CS31 Homework 5: x86_64 Loops and Functions
Due at 11:59pm Thursday, October 23, 2025

Full names of all students who worked on this:

Question 1

Convert the following C code fragment to equivalent x86_64 assembly code in two steps:
(1) First, translate the loop to its equivalent C goto version
(2) Next, translate your C goto version to x86_64, assuming that `fox` is at `%rbp - 8`, `emu` is at `%rbp - 16`, and `owl` is at `%rbp - 24`.
You must show both steps (1) and (2), and to receive partial credit annotate your x86_64 code with comments describing which part of the C code you are implementing.

```
long fox, emu, owl;
fox = 12;
emu = 90;
owl = fox - emu;
while (fox < emu) {
    fox *= 2;
    owl += fox;
}
```

```
(1) C goto version
------------------
```

```
(2) x86_64 Translation
 --------------------
```

Question 2

Trace through the following x86_64 code. Show the contents of the given memory and registers **just before the instruction at point A is executed**. Assume the `addq` instruction in `main` that is immediately after the `callq` instruction is at memory address `0x1234`. Hints:

- remember to start execution in `main`.
- `%rsp` points to the item on the top of the stack: a `push` grows the top of the stack and inserts the pushed value. A `pop` copies the value on top of the stack, then shrinks the stack.
- The sequence of instructions `leaveq`; `retq` is equivalent to the sequence: `movq %rbp, %rsp`; `popq %rbp`; `popq %rip`.

```
func:
    pushq  %rbp
    movq   %rsp, %rbp
    subq   $16, %rsp
    movq   %rdi, %rax
    addq   %rax, %rax
    movq   %rax, -8(%rbp)
    movq   -8(%rbp), %rax
    leaveq                  # point A
    retq
main:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $16, %rsp
    movq    $6, -8(%rbp)
    movq    -8(%rbp), %rdi
    callq   func
    addq    $8, %rsp        # at addr 0x1234
    movq    %rax, -8(%rbp)
    movq    $0, %rax
    leaveq
    retq
```

register	initial value	value at point A
%rax	2	
%rdi	3	
%rsp	0x88d8	
%rbp	0x88f8	

memory address	value at point A
0x8880	
0x8888	
0x8890	
0x8898	
0x88a0	
0x88a8	
0x88b0	
0x88b8	
0x88c0	
0x88c8	
0x88d0	
0x88d8	
0x88e0	
0x88e8	
0x88f0	
0x88f8	