Homework 2

CS33 Summer 2020

Due to July 21st, Tuesday 11:59PM Submit your homework on CCLE

Question1.

Switch statements. The following problem tests your understanding of switch statements that use jump tables.

Consider a switch statement with the following implementation. The code uses this `jmpq` instruction to index into the jump table:

```
0x40047b      jmpq   *0x400598(,%rdi,8)
```

Using GDB we extract the jump table:

```
0x400598:    0x0000000000400488        0x0000000000400488
0x4005a8:    0x000000000040048b        0x0000000000400493
0x4005b8:    0x000000000040049a        0x0000000000400482
0x4005c8:    0x000000000040049a        0x0000000000400498
```

Here is the assembly code for the switch statement:

```
#on entry :  %rdx = c and %rsi = b
 0x400474 :    cmp    $0x7,%edi
 0x400477 :    ja     0x40049a
 0x400479 :    mov    %edi,%edi
 0x40047b :    jmpq   *0x400598(,%rdi,8)
 0x400482 :    mov    $0x15213,%eax
 0x400487 :    retq
 0x400488 :    sub    $0x5,%edx
 0x40048b :    lea    0x0(,%rdx,4),%eax
 0x400492 :    retq
 0x400493 :    mov    $0x2,%edx
 0x400498 :    and    %edx,%esi
 0x40049a :    lea    0x4(%rsi),%eax
 0x40049d :    retq
```

Fill in the C code implementing this switch statement:

```
int main(int a, int b, int c){
     int result = 4;

     switch(a){
        case 0:

        case 1:

             c -= 5______;

        case _2_:

             result = 4 * c_;

             break;

        case _5_:

             result = ___86547___;

             break;

        case 3:

             ___c = 2____;

        case 7:

             ___b &= c___;

        default:

             result = b + 4;

    }

    return result;
}
```

Question2.

(Condition Codes and Jumps)

Address	Value	Register	Value
0x104	0x34	%rax	0x104
0x108	0xCC	%rcx	0x5
0x10C	0x19	%rdx	0x3
0x110	0x42	%rbx	0x4

Assume the addresses and registers given in the table above. Does the following code result in a jump to `.L2`? Why or why not?

```
leaq (%rax, %rbx), %rdi
cmpq $0x100, %rdi
jg .L2
```

Yes, this sequence of instructions results in a jump to .L2.
First, the load effective address loads 0x104 + 0x4 into %rdi.
%rdi now contains the value 0x108. This is compared to the constant value 0x100, which it is greater than. This sets the condition codes to reflect that %rdi is greater, and the conditional jump will read these and make the jump to .L2.

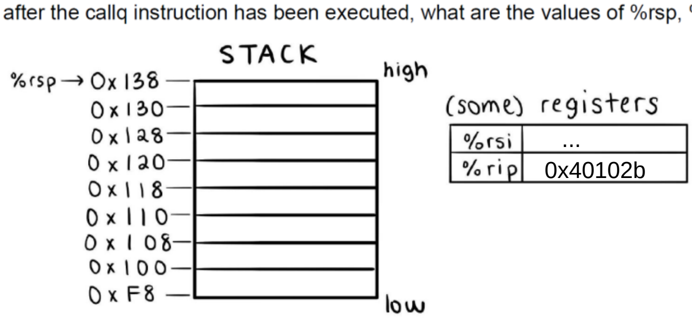
Question3.

Consider the following disassembled function:

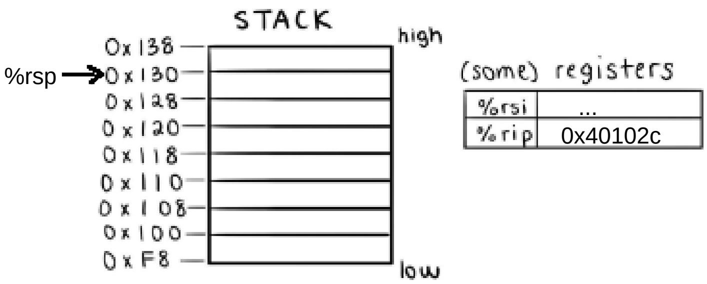
```
000000000040102b <phase_2>:
  40102b:    55                    push   %rbp
  40102c:    53                    push   %rbx
  40102d:    48 83 ec 28           sub    $0x28,%rsp
  401031:    48 89 e6              mov    %rsp,%rsi
  401034:    e8 e3 03 00 00        callq  40141c <read_six_numbers>
  401039:    83 3c 24 01           cmpl   $0x1,(%rsp)
  …
```

i) Assume %rsp initially has a value of 0x138. Draw the stack (see example diagram below) for the execution of <phase_2>, updating the stack and register values as necessary after each line is executed.

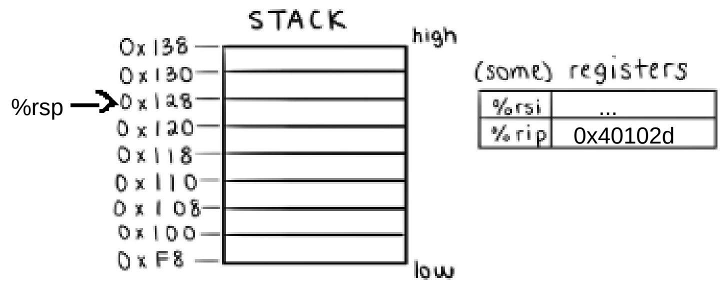
ii) Right after the callq instruction has been executed, what are the values of %rsp, %rsi, and %rip?



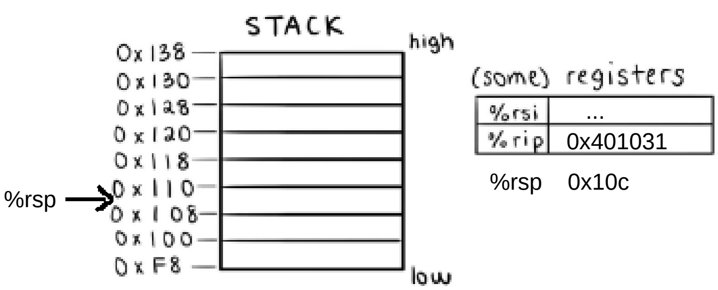
push %rbp



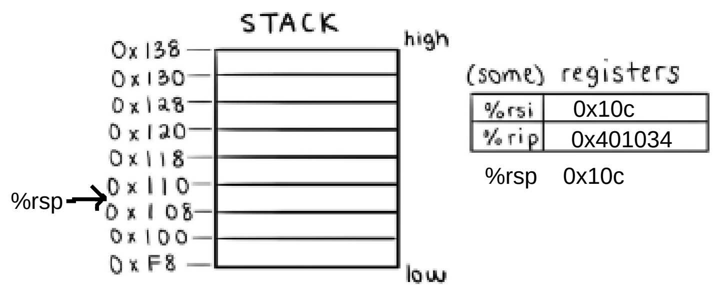
push %rbx



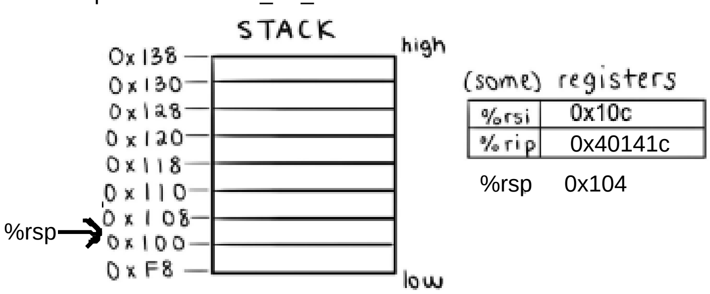
sub $0x28, %rsp



mov %rsp, %rsi



callq 40141c <read_six_numbers>



ii)

We can see that the values after the procedure call are:

%rsp - 0x10c

%rsi - 0x40141c

%rsp - 0x104