CMPT 295: Assembly Language Programming

The following questions are sample questions based on using the x86-64 assembly language:

1. Write an assembly language program segment that will calculate 5 + 4 - 6 if the values 5, 4, and 6 are stored in locations 0x40000C, 0x4000E, and 0x40010 respectively. The result is left in a register.

2. Consider the following program:

```
        mov   $0x78, %rax
LOOP:   sub   $1, %rax
        cmp   $0, %rax
        jne   LOOP
        . . .
```

   (a) How many times is the **sub** instruction executed?

   (b) How many times does the program branch to LOOP?

   (c) What change is necessary to make the program branch 210 times?

3. Assume a subprogram is already defined for computing the square root of an integer value. This subprogram is named SQRT and requires a 4 byte integer as its argument. It returns the result as 8 bytes The first four bytes define the digits, as an unsigned integer, to the left of the decimal, the second four bytes define the digits to the right of the decimal point, again as an unsigned integer. For example the square root of 2 (i.e., 1.4142135) would be returned as 1 and 4142135. Write a calling program segment to call the subprogram, SQRT, and then check if the argument is a perfect square. Store the value 1 at a location labelled "PERFECT" if the number is a perfect square; else store 0 there.

4. Demonstrate your understanding of addressing modes by providing three different instructions that will place the value 0 in register `%rax`. You may assume that the value 0 is available in external memory location "ZERO" and also in register `%rdi`.

5. Suppose the **neg** instruction does not exist. Write an assembly language subprogram, `NEG`, to compute the negation of a 64-bit number. The subprogram uses the call by value method to pass arguments and results.

6. A subprogram requires the use of four registers `r12, r13, r14`, and `r15`. As well, it will need to allocate three quad values to store temporary results obtained during execution. Construct a stack frame diagram showing how stack memory should be organized for proper execution

Also check out the following problems from the textbook: 3.1, 3.2, 3.8, 3.9, 3.18, 3.34, 3.35