# Computer Science 304
# Computer Organization
# Fall 2020
# Assignment 7

**Due: Friday, 11/6/2020 11:59 p.m.**

Answer the following questions and submit solutions at the beginning of class on the due date. Show your work for full credit. All submissions must be completely your own work.

1. [16 points] Given the following memory, register, and immediate values, state the value of the operand.

| Address | Value |
|---|---|
| 0x100 | 0x03 |
| 0x104 | 0xCD |
| 0x108 | 0x10 |
| 0x10C | 0x02 |

| Register | Value |
|---|---|
| %eax | 0x100 |
| %ecx | 0x02 |
| %edx | 0x01 |

   a. **`%eax`**
   b. **`$0x104`**
   c. **`0x104`**
   d. **`(%eax)`**
   e. **`4(%eax)`**
   f. **`(%eax,%ecx,4)`**
   g. **`8(%eax,%edx,4)`**
   h. **`0xFF(%edx,%ecx,2)`**

2. [10 points] For each of the following, state the equation that the instruction represents. The first result has been done for you.

| Assume: %eax = x; %ecx = y; %edx = z; | |
|---|---|
| **INSTRUCTION** | **RESULT** |
| leal 6(%eax), %edx | z = 6 + x |
| leal 1(%ecx), %ecx | (a) |
| leal (%edx, %ecx, 8), %eax | (b) |
| leal 8(%edx, %edx, 2), %ecx | (c) |
| leal 0xA(, %ecx, 8), %eax | (d) |
| leal 7(%eax, %edx, 4), %ecx | (e) |

3. [20 points] Consider the C function listed below.

```
void mystery (int a, int b) {
   int i, flag;

   while (a <= b) {
      flag = 0;

      for (i = 2; i <= a / 2; ++i) {
         if (a % i == 0) {
            flag = 1;
            break;
         }
      }

      if (flag == 0)
         printf("%d ", a);

      ++a;
   }
}
```

a. [3 points] Write a clear statement as to what this function actually does (not how it is implemented, but its purpose when called).
b. [2 points] What do the variables **a** and **b** represent?
c. [1 point] If this function were called as **mystery (10, 50)**, what would **a**'s value be at the very end of the function, after the **while** loop, right before returning?
d. [5 points] Rewrite the entire function with **a** and **b** passed by reference.
e. [4 points] If we declared **int** variables **c** and **d** in the calling function, and initialized them to **10** and **50**, respectively, how would **mystery** be called using the modified code from (d)? What would be the values of **c** and **d** upon return? Is passing by reference a good idea for **a** or **b**? Why or why not?
f. [3 points] The **for** loop contains some unnecessary iterations. Rewrite the **for** statement without unnecessary iterations or redundant computation.
g. [2 points] Considering what you know about variable **i**, is it assigned the right type? That is, can it be declared more narrowly to match its possible values?

4. [14 points] Fill in the blanks of the following C code, given the equivalent Y86 assembly code:

```
long test (long x, long y, long z)  {
   long val = ____(a)____;

   if (____(b)____) {
      if (____(c)____)
         val = ____(d)____;
      else
         val = ____(e)____;
   }
   else if (____(f)____)
      val = ____(g)____;

   return val;
}
```

```
# assume x in %edi, y in %esi, and z in %edx
.pos 0x100
test:   rrmovl %edi, %eax
        irmovl $1,   %ebx
        subl   %ebx, %eax
        rrmovl %edi, %ebx
        addl   %edx, %ebx
        subl   %esi, %ebx
        jge    L1
        irmovl $2,   %ebx
        subl   %edx, %ebx
        jg     L2
        rrmovl %esi, %ebx
        addl   %esi, %ebx
        addl   %ebx, %eax
        jmp L4

L2:     subl   %edi, %eax
        jmp L4

L1:     rrmovl %esi, %ebx
        addl   %edx, %ebx
        subl   %edi, %ebx
        jge    L4
        addl   %edx, %eax

L4:     ret
```

5. [20 points] Translate the Y86 code in problem 4 to byte code. Show each memory address location beside the corresponding byte sequence.

6. [20 points] Decode the following byte sequences into Y86 32-bit instructions. Hint: Repeated memory addresses indicate assembler directives and labels. The first **0x000** address indicates a **.pos** directive; the other addresses are labels. Use labels **L1** and **L2** for those locations.

```
0x000:
0x000: 30f000000000
0x006: 30f101000000

0x00c:
0x00c: 30f701000000
0x012: 6217
0x014: 731b000000
0x019: 6010

0x01b:
0x01b: 30f201000000
0x021: 6021
0x023: 30f264000000
0x029: 6112
0x02b: 750c000000

0x030: 00
```

What does this program do?