CMPE 12
HW #3
Robert Loquinario
rloquina@ucsc.edu

1. (15 pts) Write an LC-3 assembly language program that counts the number of 1s in the value stored in R0 and stores the result in R1. For example, if R0 contains 0001001101110000, then R1 should store the value 6.

Answer:

```
        ;R1 will be my counter
        ;R2 will be the loop count
        ;R3 will use this as a temp to copy R0

        .ORIG x3000             ;Start point
        AND R1, R1, 0           ;clear
        AND R2, R2, 0
        AND R3, R3, 0
        AND R3, R0, 0           ;copy R0 to R3
        AND R3, R3, 0           ;add 0 to check highest bit (neg or pos)
        BRZP FOO                ;go to FOO if greater than 0

FOO
        ADD R2, R2, -15         ;this is the counter, looping 15 times

LOOP1
        BR    LOOP2             ;if >=, go to LOOP2
        ADD R1, R1, 1           ;count
LOOP2
        ADD R2, R2, 1           ;increment the counter
        BRn LOOP1               ;if not done, go back, other wise exit

HALT

.END
```

2. (10 pts) The following program adds the values stored in memory locations A, B and C, and stores the result into memory. There are two errors in

the code. For each, describe the error and indicate whether it will be detected at assembly time or run time.

```
.ORIG x3000
ONE LD R0, A
ADD R1, R1, R0
TWO LD R0, B
ADD R1, R1, R0
THREE LD R0, C
ADD R1, R1, R0
ST R1, SUM
HALT
        A .FILL x0001
        B .FILL x0002
        C .FILL x0003
        D .FILL x0004
        .END
```

ANSWER:

1. The first mistake is that SUM is not defined/has no label and will get detected in the first pass in assembly time.
2. The second mistake is that R1 is not cleared or defined.

3. (5 pts) Name some of the advantages of doing I/O through a TRAP routine instead of writing the routine yourself each time you would like your program to perform I/O.

ANSWER: It makes coding easier, since we don't have to worry about what register is being used for input and output data.

4. (5 pts)

1. **How many trap service routines can be implemented in the LC-3? Why?**
2. **How many accesses to memory are made during the processing of a TRAP instruction?**

ANSWER:

a. Trap vectors are 8 bits wide therefore 256 trap routines can be implemented.
b. There is only 1 memory accessing since we get the starting address by accessing memory location by extending trap vectors.

5. **(20 pts) Consider the following LC-3 assembly language program:**

```
.ORIG x3000
L1 LEA R1, L1
AND R2, R2, 0
ADD R2, R2, 2
LD R3, P1
L2 LDR R0, R1, xC
OUT
ADD R3, R3, -1
BRz GLUE
ADD R1, R1, R2
BR L2
GLUE HALT
P1 .FILL xB
.STRINGZ "HBoeoakteSmtHaotren!s"
.END
```

1. **After this program has run, what binary pattern is stored in memory location x3005**
2. **Which instruction (provide memory address) is executed after instruction x3005 is executed?**
3. **Which instruction (provide mem add.) is executed prior to instruction x3006**
4. **What is the output of this program?**

ANSWER:

1. The TRAP instruction OUT is executed with the binary pattern: 1111 0000 0010 0001 in location x3005
2. Using the LC3 simulator, the instruction ST R7, x043B is executed with the memory address of x0430
3. According to LC3 simulator, the instruction RET is executed with the memory address of x0437
4. "HookemHorns"

6. **(30 pts) Perform the following multiplications in 4-bit 2SC and provide both the binary and decimal answers.**

```
1. 5x6:        5 = 0101, 6 = 0110 ->       0101
                                          x0110
                                  ----------------
                                           0000
                                          01010
                                         010100
                                        0000000
                                  --------------
                                       00011110 = #30
2.  -6 x 4 -> -6 = 1010, 4=0100 ->         1010
                                         x 0100
                                      ----------
                                           0000
                                          00000
                                         101000
                                        0000000
                                     -----------
                                       00101000 = -24
3.  2x6 -> 2= 0010,6=0110 ->               0010
                                          x0110
                                  --------------
                                           0000
                                          00100
                                         001000
                                        0000000
                                  -------------
                                        0001100 = 12
4.  -3 x -2 =                              0011
     3 x 2 -> 3=0011, 2=0010              x0010
                                       --------
                                           0000
                                          00110
                                         000000
                                        0000000
                                        0000110 = 6
```

5. -8 x 7 -> 8 x -7

8= 00001000

-7=11111001

```
      11111001
     x00001000
     ------------
      00000000
     000000000
    0000000000
   11111001000
  -----------
   ----11001000 = -56
```