James Vrionis
CMPE-12L

HW #3

1. 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.

```
            .ORIG   x3000
            AND     R5, R5, #0
            ADD     R5, R5, #1  ;R5 will act as a mask to
                                ;mask out the unneeded bit
            AND     R1, R1, #0  ;zero out the result registers
            AND     R2, R2, #0  ;R2 will act as a counter
            LD      R3, NegSixt
MskLoop     AND     R4, R0, R5 ;mask off the bit
            Brz     NotOne      ;if bit is one increment
                                ;increment the result
NotOne      ADD     R5, R5, R5 ;shift the mask one bit left
            ADD     R2, R2, #1 ;increment counter (tells us
                                ;where we are in bit pattern)
            ADD     R6, R2, R3
            BRn     MskLoop    ;not done yet go back and
                                ;check other bits
            HALT
NegSixt     .FILL   #-16
            .END
```

2. 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.

```
1.          .ORIG x3000
2. ONE       LD R0, A
3.          ADD R1, R1, R0
4. TWO       LD R0, B
5.          ADD R1, R1, R0
6. THREE     LD R0, C
7.          ADD R1,R1,R0
8.          ST R1, SUM
9.          HALT
10. A       .FILL x0001
11. B       .FILL x0002
12. C       .FILL x0003
13. D       .FILL x0004
14.         .END
```

(2 Continued)
Error 1: Line 8: ST R1, SUM
SUM is an undefined label. This error will be detected at assembly time.
Error 2: Line 3: ADD R1, R1, R0
R1 was not initialized before it was used; therefore, the result of this ADD instruction may be correct. This error will be detected at run time.

3. 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.

The most important advantage of doing I/O through a tap routine is the fact that it is not necessary for the programmer to know the gory low-level details of the specific hardware's input/output mechanism. These details include:

- the hardware data registers for the input and output devices
- the hardware status registers for the input and output devices
- the asynchronous nature of the input relative to the executing program.

Besides, these details may change from computer to computer. The programmer would have to know these details for the computer he's working on in order to be able to do input/output. Using a trap routine requires no hardware-specific knowledge on part of the programmer and saves time.

4.

1. How many trap service routines can be implemented in the LC-3? Why?

   In LC3 we have 256 TRAP service routines as there are 256 vector locations. These locations are located at x0 to xFF in memory.

2. How many accesses to memory are made during the processing of a TRAP instruction?

   One access to memory is made during the process of TRAP instruction which is already in IR.

James Vrionis
CMPE-12L

5. Consider the following LC-3 assembly language program:

```
1)          .ORIG x3000
2)  L1      LEA R1, L1
3)           AND R2, R2, 0
4)          ADD R2, R2, 2
5)          LD R3, P1
6) L2       LDR R0, R1, xC
7)          OUT
8)          ADD R3, R3, -1
9)          BRz GLUE
10)         ADD R1, R1, R2
11)         BR L2
12) GLUE    HALT
13) P1      .FILL xB
14)         .STRINGZ "HBoeoakteSmtHaotren!s"
15)         .END
```

1. After this program has run, what binary pattern is stored in memory location x3005?
The binary pattern stored in the memory location x3005 is 111 000 0010 0001 because the instruction TRAP OUT or TRAP x21 is executed which write a character in R0[7:0] to the console display. Instruction format is shown below:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRAP		TrapVect8													
1	1	1	1	0	0	0	0	0	0	1	0	0	0	0	1

∴ the binary pattern stored in x3005 is 1111 0000 0010 0001

2. Which instruction (provide a memory address) is executed after instruction x3005 is executed?

The instruction OUT or TRAP x21 is executed after instruction x3005. The associated memory address for the instruction OUT is x0430.

3. Which instruction (provide a memory address) is executed prior to instruction x3006

The instruction RET is executed prio to instruction x3006. The memory address associated to this instruction is x0437. The reason why is once the instrunction OUT or TRAP x21 is executed the compiler should return the control to the next instructions and this is done by having RET instruction. In short, it returns controll to the next instruction to be executed.

4. What is the output of this program?
HookemHorns
--- Halting the processor ---
1- Initially immediate value 2 is stored in Register R2
2- Load hexadecimal value 0B to Register R3 to display 11 characters from the give string.
3- Load the effective address of R1.
4- Display the letter present in effective address.
5- Decrement the value of R3, if contents of R3 is not equal to zero, add 2 to R2 to get the next effective address.
6- Jump to step 3 and repeat the process
7- if R3 is equal to zero, Halt the program.

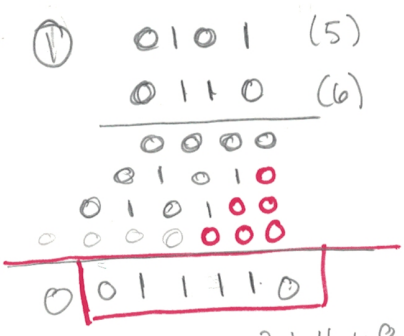
6. Perform the following multiplications in 4-bit 2SC and provide both the binary and decimal answers.

				2SC Binary		*Decimal*
1. 5 x 6	=>	0101 x 0110	=>	0011110	=>	30
2. -6 x 4	=>	0101 x 0100	=>	11101000	=>	-24
3. 2 x 6	=>	0010 x 0110	=>	0001100	=>	12
4. -3 x -2	=>	1101 x 1110	=>	0000110	=>	6
5. -8 x 7	=>	1000 x 0111	=>	1001000	=>	-56

(work for 6 on next page)

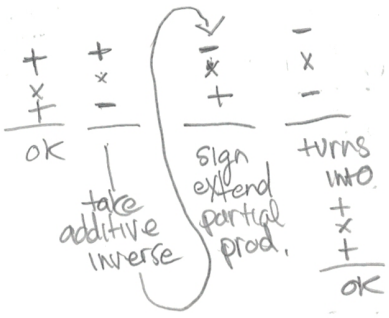
Work

①

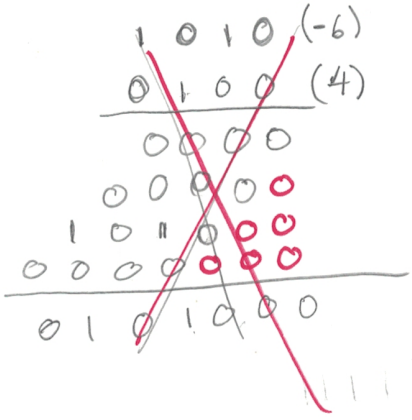


2+4+8+16 = 30

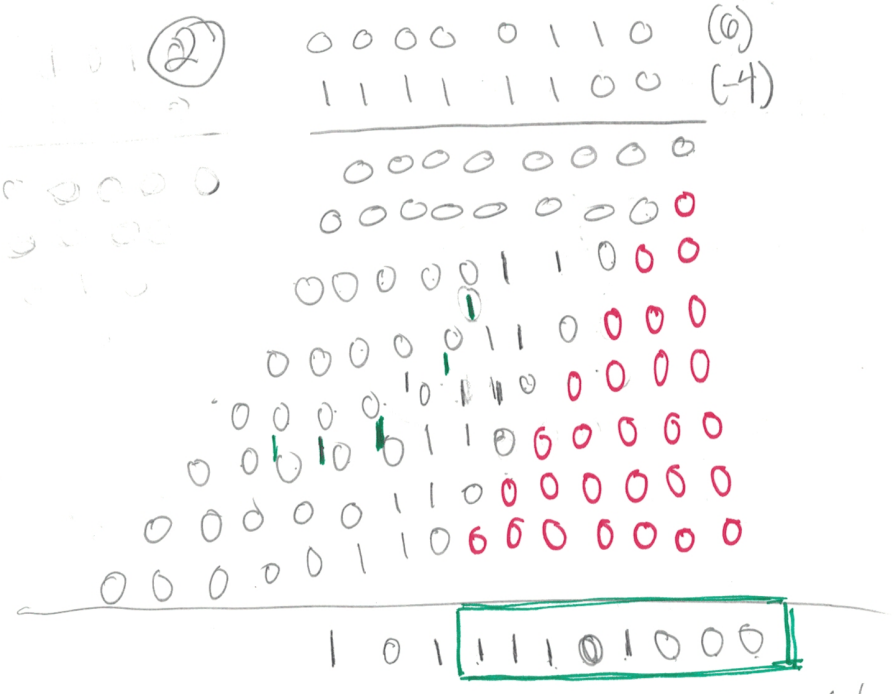
10+16+4 =



2)



②



128 − 8+32+64

128 − 104 = −24

Work 2

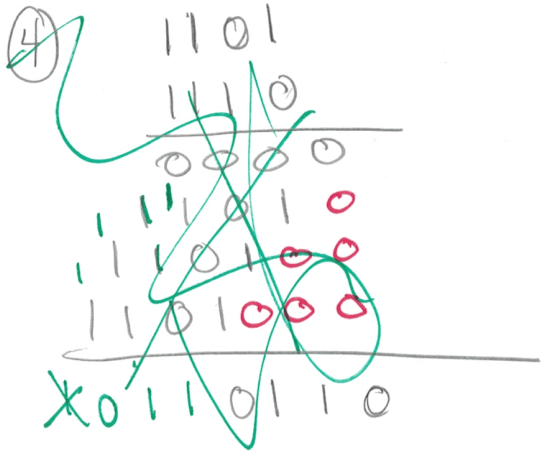
③

0010 (2)
0110 (6)

0000
0010
0010
0000
00 0110

4+8 = 12

④



④

1101 (-3)
1110 (-2)

10000
11010
11101
11010
1011 0110

4+2 = 6

-12+6 = -6

④

0011 (3)
0010 (2)

0000
0011
0000
0000
00 0110 = 6

⑤

01000 (8)
11001 (-7)

01000
00000
00000
01000
01000
01 0010 00

-128+64+8 = -128+72 = -56