Chapter 7&8

1. consider the following program written in LC-3 assembly language:

```
                .ORIG x3000
                AND R5, R5, #0
                LEA R0, ARRAY
                LD  R1,  N
                LDR R2,  R0,   #0
                NOT R2, R2
                ADD R2, R2,  #1
LOOP            LDR  R3,  R0,  #0
                ADD  R3,  R3,  R2
                BRnp   DONE
                ADD  R0,  R0, #1
                ADD  R1,  R1,  #-1
                BRp LOOP
                ADD    R5,    R5,  #1
DONE            ST   R5,  OUTPUT
                HALT
ARRAY           .BLKW  #20
N               .FILL   #20
OUTPUT          .BLKW  #1
                .END
```

What must be the case for 1 to be stored in OUTPUT? Answer in 15 words or fewer.

When all elements in array are same.

2.An Aggie tried to write a recursive subroutine which, when given an integer n, return the sum of the first n positive integers. For example, for n = 4, the subroutine returns 10 (i.e., 1 + 2 + 3 + 4). The subroutine takes the argument n in R0 and returns the sum in R0.

```
SUM     ADD R6, R6, #-1
        STR R7, R6, #0
        ADD R6, R6, #-1
        STR R1, R6, #0
        ADD R1, R0, #0
        ADD R0, R0, #-1
        JSR SUM
        ADD R0, R0, R1
        LDR R1, R6, #0
        ADD R6, R6, #1
        LDR R7, R6, #0
        ADD R6, R6, #1
        RET
```

Unfortunately, the recursive subroutine does not work. What is the problem? Explain in 15 words or fewer.And modify the program to make it work.

There is no base case.

```
SUM     ADD R6, R6, #-1
        STR R7, R6, #0
        ADD R6, R6, #-1
        STR R1, R6, #0
        ADD R1, R0, #0
        ADD R0, R0, #-1
        BRnz NEXT
        JSR SUM
NEXT    ADD R0, R0, R1
        LDR R1, R6, #0
        ADD R6, R6, #1
        LDR R7, R6, #0
        ADD R6, R6, #1
        RET
```

3.Memory locations x5000 to x5FFF contain 2's complement integers. What does the following program do?

```
.ORIG x3000
 LD  R1, ARRAY
 LD  R2, LENGTH
 AND R3, R3, #0
```

```
AGAIN   LDR R0, R1, #0
        AND R0, R0, #1
        BRz SKIP
        ADD R3, R3, #1
SKIP    ADD R1, R1, #1
        ADD R2, R2, #-1
        BRp AGAIN
        HALT
ARRAY  .FILL x5000
LENGTH .FILL x1000
       .END
```

Please write your answer in the box below. Your answer must contain at most 15 words. Any words after the first 15 will NOT be considered in grading this problem.

count the number of odd numbers in the array.

4.It is easier to identify borders between cities on a map if a adjacent cities are colored with the different colors. For example, in a map of Texas, one would not color Austin and Pflugerville with the same color, since dong so would obscure the border bewteen the two cities.

Shown below is the recursive subroutine EXAMINE. EXAMINE examines the data structure representing a map to see if any pair of adjacent cities have the same color. Each node in the data structure contains the city's color and the addresses of the cities it borders. If no pair of adjacent cities have the same color, EXAMINE returns the value 0 in R1. If at least one pair of adjacent cities have the same color, EXAMINE returns the value 1 in R1. The main program supplies the address of a node representing one of the cities in R0 before executing JSR EXAMINE.

```
        .ORIG x4000
EXAMINE ADD R6, R6, #-1
        STR R0, R6, #0
        ADD R6, R6, #-1
        STR R2, R6, #0
        ADD R6, R6, #-1
        STR R3, R6, #0
        ADD R6, R6, #-1
        STR R7, R6, #0

        AND R1, R1, #0     ; Initialize output R1 to 0
        LDR R7, R0, #0
        BRn RESTORE        ; Skip this node if it has already been visited
```

```

        LD  R7, BREADCRUMB
        STR R7, R0, #0     ; Mark this node as visited
        LDR R2, R0, #1     ; R2 = color of current node
        ADD R3, R0, #2

AGAIN   LDR R0, R3, #0     ; R0 = neighbor node address
        BRz RESTORE
        LDR R7, R0, #1
        NOT R7, R7         ; <-- Breakpoint here
        ADD R7, R7, #1
        ADD R7, R2, R7     ; Compare current color to neighbor’s color
        BRz BAD
        JSR EXAMINE        ; Recursively examine the coloring of next neighbor
        ADD R1, R1, #0
        BRp RESTORE        ; If neighbor returns R1=1, this node should return R1=1
        ADD R3, R3, #1
        BR  AGAIN          ; Try next neighbor

BAD     ADD R1, R1, #1
RESTORE LDR R7, R6, #0
        ADD R6, R6, #1
        LDR R3, R6, #0
        ADD R6, R6, #1
        LDR R2, R6, #0
        ADD R6, R6, #1
        LDR R0, R6, #0
        ADD R6, R6, #1
        RET

BREADCRUMB .FILL x8000
        .END
```

Your job is to construct the data structure representing a particular map. Before executing JSR EXAMINE, R0 is set to x6100 (the address of one of the nodes), and a breakpoint is set at x4012. The table below shows relevant information collected each time the breakpoint was encountered during the running of EXAMINE.

PC	R0	R2	R7
x4012	x6200	x0042	x0052
x4012	x6100	x0052	x0042
x4012	x6300	x0052	x0047
x4012	x6200	x0047	x0052
x4012	x6400	x0047	x0052
x4012	x6100	x0052	x0042
x4012	x6300	x0052	x0047
x4012	x6500	x0052	x0047
x4012	x6100	x0047	x0042
x4012	x6200	x0047	x0052
x4012	x6400	x0047	x0052
x4012	x6500	x0052	x0047
x4012	x6400	x0042	x0052
x4012	x6500	x0042	x0047

Construct the data structure for the particular map that corresponds to the relevant information obtained from the break- points. Note: We are asking you to construct the data structure as it exists AFTER the recursive subroutine has executed.

x6100	x8000	x6300	X8000	x6500	X8000
x6101	X0042	x6301	X0047	x6501	X0047
x6102	X6200	x6302	x6200	x6502	X6100
x6103	X6400	x6303	x6400	x6503	X6200
x6104	x6500	x6304	X0000	x6504	X6400
x6105	X0000	X6305		x6505	X0000
x6106		x6306		x6506	
x6200	X8000	x6400	x8000		
x6201	X0052	x6401	x0052		
x6202	X6100	x6402	x6100		
x6203	X6300	x6403	x6300		
x6204	X6500	x6404	x6500		
x6205	X0000	x6405	X0000		
x6206		x6406			

5. The following program, after you insert the two missing instructions, will examine a list of positive integers stored in consecutive sequential memory locations and store the smallest one in location x4000. The number of integers in the list is contained in memory location x4001. The list itself starts at memory location x4002. Assume the list is not empty (i.e., the contents of x4001 is not zero.)

```
                .ORIG x3000
                LDI R1, SIZE
                LD  R2, LISTPOINTER
                LDR R0, R2, #0
                ADD R1, R1, #-1
                BRz ALMOSTDONE        ;Only one element in the list
AGAIN           ADD R2,R2,#1

                LDR R3,R2,#0
                NOT R4,R3
                ADD R4,R4,#1
                ADD R4,R0,R4
                BRnz  SKIP
                ADD R0,R3,#0
```

```
SKIP            ADD R1,R1,#-1
                BRp AGAIN

ALMOSTDONE      LD R5,MIN
                STR R0,R5,#0
                HALT

MIN             .FILL x4000
SIZE            .FILL x4001
LISTPOINTER     .FILL x4002
                .END
```

Your job: Insert the two the missing instructions.

6.Your job in this problem will be to add the missing instructions to a program that detects palindromes. Recall a palin- drome is a string of characters that are identical when read from left to right or from right to left. For example, racecar and 112282211. In this program, we will have no spaces and no capital letters in our input string – just a string of lower case letters.

The program will make use of both a stack and a queue. The subroutines for accessing the stack and queue are shown below. Recall that elements are PUSHed (added) and POPped (removed) from the stack. Elements are ENQUEUEd (added) to the back of a queue, and DEQUEUEd (removed) from the front of the queue.

```
          .ORIG x3050
PUSH      ADD R6, R6, #-1
          STR R0, R6, #0
          RET
POP       LDR R0, R6, #0
          ADD R6, R6, #1
          RET
STACK     .BLKW #20
          .END


          .ORIG x3080
ENQUEUE   ADD R5, R5, #1
          STR R0, R5, #0
          RET
DEQUEUE   LDR R0, R4, #0
          ADD R4, R4, #1
          RET
```

```
QUEUE       .BLKW #20
            .END
```

The program is carried out in two phases. Phase 1 enables a user to input a character string one keyboard character at a time. The character string is terminated when the user types the enter key (line feed). In Phase 1, the ASCII code of each character input is pushed on a stack, and its negative value is inserted at the back of a queue. Inserting an element at the back of a queue we call enqueuing.

In Phase 2, the characters on the stack and in the queue are examined by removing them, one by one from their re- spective data structures (i.e., stack and queue). If the string is a palindrome, the program stores a 1 in memory location RESULT. If not, the program stores a zero in memory location RESULT. The PUSH and POP routines for the stack as well as the ENQUEUE and DEQUEUE routines for the queue are shown below. You may assume the user never inputs more than 20 characters.

The program for detecting palindromes (with some instructions missing) .

Your job is to fill in the missing instructions.

```
            .ORIG X3000
            LEA   R4, QUEUE
            LEA   R5, QUEUE
            ADD   R5, R5, #-1
            LEA   R6, ENQUEUE       ;Initialize SP
            LD    R1, ENTER
            AND   R3, R3, #0
;
            LEA R0,PROMPT
            TRAP x22
PHASE1      TRAP x20
            ADD R2,R0,R1
            BRz PHASE2
            JSR PUSH
            NOT R0,R0
            ADD R0,R0,#1
            JSR ENQUEUE
            ADD R3, R3, #1
            BRnzp PHASE1
;
PHASE2      JSR POP
            ADD R1,R0,#0
            JSR DEQUEUE
            ADD R1, R0, R1
            BRnp  FALSE
            ADD R3,R3,#-1
            BRz TURE
            BRnzp PHASE2
;
TRUE        AND R0, R0, #0
```

```
             ADD R0, R0, #1
             ST  R0, RESULT
             HALT
FALSE        AND R0, R0, #0
             ST  R0, RESULT
             HALT
RESULT       .BLKW #1
ENTER        .FILL x-0A
PROMPT       .STRING "Enter an input string"
             .END
```

More problems approaching!


7850
75
75
75
125
25
25
0
25
150
50
Menu
More zombies approaching!