ENGR 273 review

Text Question 5.10

1110 001 100100011	LEA R1,0x123	R1 = 0x3123
0010 010 100100011	LD R2,0x123	R2 = Mem[0x3123] = 0x4567
0110 011 010 000001	LDR R3,R2,0x01	R3 = Mem[0x4567 + 1] = 0xFED3
1010 100 100100011	LDI R4,0x123	R4 = Mem[Mem[0x3100]] = 0xABCD
1111 0000 0010 0101	TRAP 0x25	

Text Question 7.1

0xA624

Text Question 11.8

1. The C preprocessor takes C source and header files as provided by the programmer.
2. The C compiler receives input from the preprocessor. The preprocessors have expanded all preprocessor macros. All files have been included at the appropriate places.
3. The linker receives one or more object modules. Possibly, these object modules have external variable and subroutine references, which the linker will try to resolve amongst the modules and possibly in object libraries.

Text Question 12.6

The output of the program will be as follows:

2
2
3
0 (or undefined or garbage)

Text Question 12.7

	expression	**a**	**b**
10.	5	5	5
11.	10	10	10
12.	3072	3072	9

Text Question 13.4

	Output/Behavior when x = 0	**Output/Behavior when x = 1**
1.	‘x does not equal 0’ is output	‘x does not equal 0’ is output
4.	y = 3 is executed as well as all other lines after the assignment before the first break is encountered	y = 4 is executed followed by the break statement

Text Question 13.11

```
#include <stdio.h>
main()  {
    int a,b;
    b = 10;
    a = 5;
    a = b – a;
    b = a + a;
    if (a > 0)
        printf(“%c”,0x31);
}
```

Text Question 14.5

The output of the program is “**2 2** ”. The variable ‘z’ in the function MyFunc() is declared within the local scope of the function. the value of ‘z’ from main() is copied, but all operations on ‘z’ in MyFunc() affect only the local copy and not the one in main().

Quiz II -- Problem 3: Programming (80 points)

Consider the following data definition of an array of data:

```
            .ORIG x4000
ArraySize   .FILL 40
ArrayA      .FILL 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
            .FILL 90,-987,232,3,82,124,-276,-111,65,77,-333,-656,123,22,44
            .FILL 34,34,76,66,87
result .BLKW 1
```

3a) (20 points) Write an algorithm to compute the sum of all of the numbers in `ArrayA`. Put the result in `result`. Then print a '+' to the console if the result is positive, and a '-' if the result is negative. **You must use `ArraySize` as the size of the array for loop counting**. Use looping and branching, and try to make your program execute as few instructions as possible. Include comments for full credit.

3c) (55 points) Write an LC-2 program to execute the algorithm above. Include EVERYTHING which is needed to run a program (.ORIG, HALT)

3b) (5 points) How many instructions will execute while running this program?

Answer

```
      .ORIG       x3000
      LDI         R5,pArraySize     ; R5 counts down size of array
      LD          R4,pArrayA        ; R4 points to array elements
      AND         R3,R3,#0          ; R3 will be the sum
ADDLP LDR         R2,R4,#0          ; R2 get next array element
      ADD         R3,R3,R2          ; add next element to R2
      ADD         R4,R4,#1          ; move array pointer
      ADD         R5,R5,#-1         ; decrement array size
      BRp         ADDLP             ; loop again, if still positive
      STI         R3,pResult        ; store sum in result
      LD          R3,R3,#0          ; test value of the sum
      BRp         PPLUS
      LD          R0,MSIGN          ; set R0 to '-' if sum is < 0
      BRnzp       PRNTT
PPLUS LD          R0,MPLUS          ; set R0 to '+' if sum is > 0
PRNT  OUT                           ; print R0 to console
      HALT
pArraySize        .FILL       x4000
pArrayA           .FILL       x4001
pResult           .FILL       x4029
MSIGN             .FILL       x2d
MPLUS             .FILL       x2b
      .END
```

This program executes 5 instructions inside the loop and 8 or 9, depending on the sum, instructions outside the loop. Since the loop is executed 40 times, the total number of instructions executed is either 208 or 209.

Points worth remembering

Because ArraySize, ArrayA, and Result are all on different pages, you must use either LDI or LDR to load their values. Similarly, you must use STI or STR to store to them.

```
; MULTIPLY,   James Conrad, 2000-10-29
: Write an algorithm for the LC-2 to Multiply two entered numbers and output
; Input:  2 numbers from the keyboard, in the range of -32768 to +32767 -
;     including 0
; Output:  One number to the console

             .ORIG x3000
             AND R7,R7,#0        ;clear R0

;===Input the first number, keep track of the "negative" product===
             TRAP x24                  ; Read first number
             BRz FINALEXT        ; if zero, no need to multiply - end
             ADD R3,R0,#0        ;   Otherwise, save
             BRp NEXTNUM               ; If Positive, no need to invert
             NOT R3,R3                 ; Compute 2's complement
             ADD R3,R3,#1
             ADD R7,R7,#1        ; Set the negative flag

;===Input the next number, keep track of the "negative" product===
NEXTNUM      TRAP x24                  ; Read next number
             BRz FINALEXT        ; if zero, no need to multiply - end
             ADD R4,R0,#0        ;   Otherwise, save
             BRp SWAP                  ; If Positive, no need to invert
             NOT R4,R4                 ; Compute 2's complement
             ADD R4,R4,#1

             ADD R7,R7,#0        ; Test the "negative flag"
             BRp TWONEG                ; Is neg * neg = pos
             ADD R7,R7,#1        ; Else Set the negative flag
             BRnzp SWAP                ;
TWONEG AND R7,R7,#0

;===Put the bigger of the numbers in R1, smaller in R2===
SWAP         NEG R2,R4                 ; Compute -R4
             ADD R2,R2,1
             ADD R2,R3,R2        ; set flags.  n means R4 was bigger
             BRn R4BIGGER
             ADD R1,R3,#0        ;else, R3 was bigger
             ADD R2,R4,#0        ;
             BRznp COMPUTE
R4BIGGER     ADD R2,R3,#0        ;
             ADD R1,R4,#0        ;

;===Compute R3 = R1 * R2 - Remember, the LC-2 has no multiply instruction===
COMPUTE      AND R3,R3,#               ; Accumulator
LOOPHERE     ADD R3,R3,R1        ; Multiply by repeatable adding
             BRn BADNUM
             ADD R2,R2,#-1       ; add "R2" times
             BRp LOOPHERE

;===Output the result - invert the number if it is supposed to be negative===
             ADD R0,R3,#0
             ADD R7,R7,#0        ; Test the "negative flag"
             BRz FINALEXT        ; Done, go print result
             NOT R0,R0                 ;   Else, invert as 2's compl number
             ADD R0,R0,#1
             BRznp FINALEXT            ;   And print

BADNUM AND R0,R0,#0        ; Clear output - overflow
FINALEXT     TRAP x26                  ; Print number
             TRAP x25
```

One more problem

How would you implement the following C function on the LC-2:

```
int fetchAdd(int &x, int y) {
      int t ;
      int t = *x
      *x = *x + y ;
      return t ;
}
```

Spaces allocated on stack

R6, #0	return value
R6, #1	return address
R6, #2	dynamic link
R6, #3	x
R6, #4	y
R6, #5	t

```
fetchAdd:
      STR   R7, R6, #1       ; store return address
      LDR   R0, R6, #3       ; R0 <- x
      LDR   R1, R0, #0       ; R1 <- *x
      STR   R1, R6, #5       ; t <- R1
      LDR   R2, R6, #4       ; R2 <- y
      ADD   R1, R1, R2       ; R1 <- R1 + R2 = *x + y
      STR   R1, R0, #0       ; *x <- R1
      LDR   R1, R6, #5       ; R1 <- t
      STR   R1, R6, #0       ; return t
      LDR   R7, R6, #1       ; load return address
      LDR   R6, R6, #2       ; load dynamic link
      RET
```

Or more efficiently

```
fetchAdd:
      STR   R7, R6, #1       ; store return address
      LDR   R0, R6, #3       ; R0 <- x
      LDR   R1, R0, #0       ; R1 <- *x
      STR   R1, R6, #0       ; return *x
      LDR   R2, R6, #4       ; R2 <- y
      ADD   R1, R1, R2       ; R1 <- R1 + R2 = *x + y
      STR   R1, R0, #0       ; *x <- R1
      LDR   R6, R6, #2       ; load dynamic link
      RET
```