LC-3 Instruction Set Architecture

CS 350: Computer Organization & Assembler Language Programming

Lab 8 due Wed Mar 23

A. Why

- The instruction set architecture describes the machine language of a computer

B. Outcomes

After this lab you should be able to

- Translate between the machine and assembler representations of an LC-3 instruction
- Informally describe the action of an LC-3 instruction

C. Written Problem [22 pts]

1. [22 = 11 * 2 pts] Rewrite the table shown here, filling in any missing parts indicated by question marks. The comment for `x7FFF` shows the level of comment needed (you can write in more detail if you like. Note the values at `x800B`, `x8000C`, and `x800D` will change; show this in their comment fields.

Addr	Assembler	Action/Comment
x7FFF	LD R0,9	R0 ← M[x8009] = 12
x8000	LD R1,9	?
x8001	LDR R2,R1,0	?
x8002	LEA R4,8	?
x8003	STR R2,R4,0	?
x8004	AND R3,R3, ?	R3 ← 0
x8005	STR R3,R4,1	?
x8006	ADD R5,R0,R0	?
x8007	STI R5,2	?
x8008	TRAP x25	HALT
x8009	.FILL 12	
x800A	.FILL x800D	
x800B	.FILL 0	?
x800C	.FILL -1	?
x800D	.FILL 18	?
x800E	.FILL 0	

D. Programming Problem [28 points]

Evaluate A Quadratic Polynomial

- You are to write a multiplication subroutine and call it multiple times (using `JSR`) to evaluate a quadratic polynomial.
- The polynomial *poly*(*x*) is represented as its three coefficients; the value X_1 is an integer; you evaluate and store $Y_1 = poly(X_1)$. For example, say we begin with

```
; poly(x) = -4*X^2 + 3*X - 5
;
POLY    .FILL    -4           ; -4*X^2
        .FILL    3            ; + 3*X
        .FILL    -5           ; - 5

; To calculate: Y1 = poly(X1)
;
X_1      .FILL    -2
Y_1      .BLKW    1
```

then you would calculate $-4 \times (-2)^2 + 3 \times (-2) - 5 = -27$ and store that in `Y_1`. (Programming hint: You can't use `X1` as a variable; the LC-3 assembler treats it as the constant `x0001`.)

In the lecture on subroutines, we saw routine that calculates `X * Y` if `Y` $\geq$ `0`; you'll need to extend it to work if `Y` < `0`.

Program Structure

Your main program can be written various ways (e.g., $((A \times x_1) \times x_1)$ vs $((x_1 \times x_1) \times A)$; in pseudocode you could have

```
main:
--------------------
R0 = A                   ; coefficient A from POLY
R1 = X_1
JSR MULT                 ; R1 = A * X_1
R0 = X_1
JSR MULT                 ; R1 = A * X_1^2
temp = R1                ; temp = A * X_1^2

R0 = B                   ; coefficient B from POLY
R1 = X_1
JSR MULT                 ; R1 = B * X_1
```



```
temp = temp + R1      ; temp = A * X_1^2 + B * X
temp = temp + C       ; add coefficient C from POLY
Y_1 = temp            ; save result
HALT
```

(Programming hint: If you keep the address of `POLY` in register *reg*, you can use `LDR R0,`*reg*,0 to set `R0` = coefficient *A*; replacing `0` by `1` or `2` gives you *B* or *C*.)

Write your program as

main program

declarations of `POLY`, `X_1`, *and* `Y_1`

`MULT` *routine*

and turn in your `*.asm` file.

Grading Guide

- **Main program**
  - 2 pts Point to polynomial
  - 4 pts Calculate $A * (X_1)^2$
  - 4 pts Calculate $B * X_1$
  - 4 pts Calculate $A * (X_1)^2 + B * X_1 + C$ and store in Y_1
- **Polynomial**
  - 2 pts Coefficients stored as 3 words (*A*, *B*, and *C* above)
  - 2 pts X_1 and Y_1 declared
- **Multiply(X,Y) routine**
  - 3 pts works for all Y (>, =, or < 0).
- **Program Structure & Comments**
  - 1 pt Name and section in comments
  - 2 pts Section comment for multiply, including register usage
  - 4 pts Sections well-organized, mnemonics, arguments, and line comments formatted and readable

