ECE 120: Introduction to Computing Spring 2019 ZJUI

due at start of section on Friday 24 May 2019

Homework 13: Assembling and Debugging LC-3

1. **Protecting Your Bits!**
   The code below is intended to load a value into R3, then multiply the value by 12. Add instructions and/or data **at the locations indicated by comments** to prevent the code from changing R0 during the multiplication step. ** For credit, do not modify the current instructions, nor insert code/data other than in those locations indicated. Also, if you insert more than two instructions at any insertion point, your will receive no credit. **

   There are several reasonable strategies. First, one can move the value in R0 to another register, and move it back into R0 later. Second, one can store the value in R0 to another memory location, and load it back into R0 later. Finally, since one knows the value, one can make a second location (other than DSTR) containing the same value, then load the value again from the new location. The edits below illustrate the first approach.

```
        .ORIG x3000

        LEA R0,DSTR
        LD R3,STARTV
        ST R3,DSTR

        ; You may add code here.
        ; Save R0 to memory or to another register.
        ST R0,SAVER0

        ADD R0,R3,R3 ; x12
        ADD R3,R0,R3
        ADD R3,R3,R3
        ADD R3,R3,R3

        ; You may add code here.
        ; Restore R0 value.
        LD R0,SAVER0

        PUTS
        ; R3 value is correct here (HALT may change it
        ; in the simulator)
        HALT
DSTR    .STRINGZ "I know how to fix it!"
STARTV  .FILL #1000

        ; You may add data here.
SAVER0  .BLKW #1

        .END
```

2. Mimicking an Assembler

Write symbol tables for each of the codes below. Your symbol table should be similar in nature to that produced by the LC-3 assembler: for each label that appears in the code, your table should list the label and associate the label with an address in LC-3 memory. For an example, see P&P Section 7.3.3, pp. 186-187, or simply generate one on your own with `lc3as` (the output ending in `.sym` is a symbol table file).

a. The program given for problem 7.5.a in Patt and Patel.

Symbol	Address
LOOP	x3003
DONE	x3007
RESULT	x3009
ZERO	x300A
M0	x300B
M1	x300C

b. The program given for problem 7.16 in Patt and Patel.

Symbol	Address
LOOP	x3003
L1	x300A
NEXT	x300B
DONE	x300D
NUMBERS	x300E

c. The program given in Homework 12 Problem 5 (**before your modifications**).

Symbol	Address
START	x3000
LOOP	x3003
BELOW	x3007

d. The program given in problem 2 above (**before your modifications**).

Symbol	Address
DSTR	x3009
STARTV	x301F

3. When Assemblers Find Errors

The code below is supposed to read a sequence of ASCII characters and print them to the monitor. The characters are stored in a sequential series of memory addresses, and the last such address contains a 0 (which should not be printed). However, the ASCII characters are stored in the low 8 bits of each location, which the high 8 bits are used for some other purpose (not known to you, and not relevant to this problem).

Prof. Lumetta has produced the following code to accomplish the stated task. Unfortunately, the code has a bug. Fortunately, the assembler will find the bug.

Your tasks are as follows:

a. Identify the bug and explain it,

The LC-3 AND instruction's immediate operand must fit into a 5-bit 2's complement field. The value "xFF" does not.

b. State in which pass (first or second) the assembler identifies the bug, and

The assembler identifies the bug in the first pass.

c. Explain how to fix the bug (**in words, not in code**).

We can load xFF into a register (say, R2) from a memory location at the start of the program, and use an AND instruction with a second register operand to perform the desired masking. We have to use .FILL xFF to create the data in memory, of course.

```
          .ORIG x3000

          LEA R1,STUFF
LOOP      LDR R0,R1,#0
          AND R0,R0,xFF
          BRz DONE
          OUT
          ADD R1,R1,#1
          BRnzp LOOP
DONE      HALT
STUFF     .FILL x7768
          .FILL xAB65
          .FILL xEA6C
          .FILL xF06C
          .FILL x976F
          .FILL x1200

          .END
```