Homework 7 - Due April 16th

First Contact TA: Ryan Johnson (rajohnso@cs.wisc.edu)

1) (**5 points**)
Write a symbol table (see pg 187) for the following program:

```
     .ORIG x3000
       AND R0, R0, #0
       ADD R2, R0, #10
       LD R1, MASK
       LD R3, PTR1
LOOP LDR R4, R3, #0
       AND R4, R4, R1
       BRz NEXT
       ADD R0, R0, #1
NEXT ADD R3, R3, #1
       ADD R2, R2, #-1
       BRp LOOP
       STI R0, PTR2
       HALT
MASK .FILL x8000
PTR1 .FILL x4000
PTR2 .FILL x5000
```

2) (**5 Points**)
Describe what the LC-3 assembly program in #1 does.

3) (**5 Points**)
Do problem 7.2 in the textbook.

4) (**15 Points**)
In this and the next homework, you will work with a TicTacToe game that runs in PennSim. In it's current state, the game is playable, however some functionality is missing. We want the game to detect when a player has been "blocked," or in other words, when one player makes a move that prevents the other player from winning. For example, if player X has two Xs in the top row and then the O player places an O in the third open space, the O player has blocked the X player. To complete this problem, you will write code, plug it into the game, and test it for correctness.

Download the TicTacToe zip file here: tictactoe.zip

The zip file contains a directory called tictactoe. Inside of the directory are three files:

•lc30s.asm: This is the LC-3 operating system discussed in the PennSim guide.
•main.asm: This is the file containing the tictactoe game. This is the file you must edit.

•tictactoe.script: This is a script file that will greatly simplify your testing of your code. See below for instructions on how to use it.

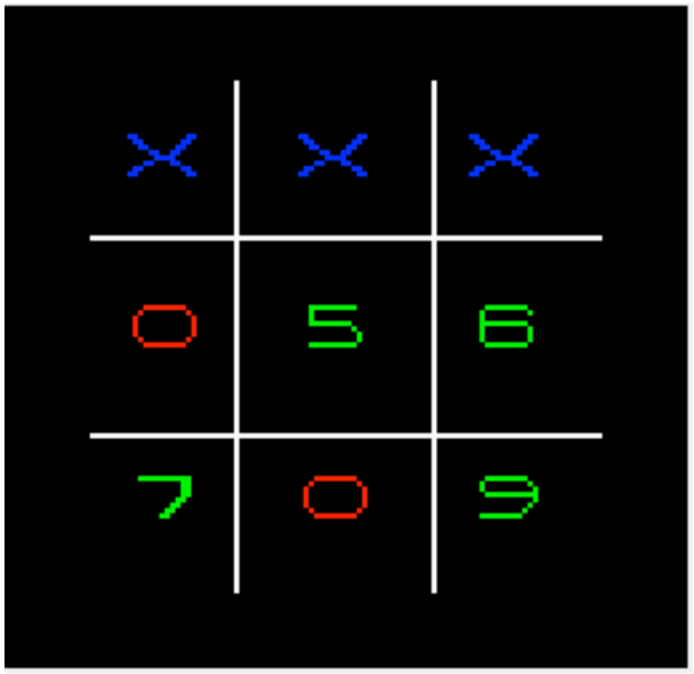
When you open main.asm in your text editor, you will see clearly marked where you should add your code. The existing code sets register R0 to 0 before executing the RET instruction. For correct functionality, you must change the existing code to do the following: If the current player has blocked the other player (i.e. stopped the other player from possible winning in their next move), you must set R0 to 1 before executing the RET instruction. If the current player's move does is not a blocking move, set R0 to 0 before executing the RET instruction.

For grading, you must submit a copy of your main.asm to the dropbox. Additionally, you must submit a screenshot of PennSim of one of the players blocking the other player. To show correct functionality in the screen shot, place a breakpoint at the RET statement as marked in the code and take the screen shot when the PC stops at that point. Make sure your include the register values in the screenshot so we can see that R0 contains 1.

The easiest way to run, debug, and test your program is to place the tictactoe directory in the same directory as PennSim. When you open up PennSim, run the following command: script tictactoe/tictactoe.script. This will reset PennSim, load the lc3os, load the main tictactoe code, and set a breakpoint at the end of the CheckBlock function. Each time you change your code, running the script will allow you to quickly rerun the game from the beginning.

Here are some more info to help you complete the problem:

Immediately before the location where you will input your code, the register R1 is loaded with the current player's most recent mark and R2 is loaded with the marks of the opposing player. In this game, marks are stored as a 9-bit code, with each of the 9 bits corresponding to one location on the board. The least significant bit corresponds to location 1. To illustrate, if the board is as follows:



Then the 9-bit code for the X player will be 000000111b, 007h. The 9-bit code for the O player will be 010001000b, 088h.

A block will only occur when there are three marks in a row (two of one type and the third and most recent being of another). There are eight possible patterns of three marks in a row TicTacToe. Since this is a relatively small number, the easiest way of checking for three marks in a row is as follows: combine the other player’s marks with the current players most recent marks in a single 9-bit code. Then compare the combined code with each of the possible blocking patterns looking for a match. If one matches, then the it is a blocking move. For example, if the X player's code is 001000011b and the O player makes the move 000000100b, combining the two results in 001000111b. If we compare this with one of the blocking move patterns, 007h, we see that there is a match. Thus we would load R0 with 1.