Homework 8 - Due at 9:55 AM on Mon, May 7th

Primary contacts for this homework: Suriyha Balaram Sankari [balaramsanka@wisc.edu] and Kamlesh Prakash [kprakash2@wisc.edu]

Instructions: You must do this homework **in groups of two**. Please hand in ONE copy of the homework in to the homework8 dropbox at Learn@UW. See the submission guidelines below.

Important Notes:

This homework must be submitted electronically to the Learn@UW dropbox. No hard copies will be accepted. Turn in problems 1-5 as a PDF document and problem 6 and problem 7 as **.asm** file (see the submission guidelines below). List the **full names, UWIDs, and sections** of all members in your group in the PDF document and also in the comment section of your submission.

Submission Guidelines:

1. Submit only **one** set of files per group to the folder **homework8**
2. List the **full names, UWIDs, and sections** of all members in your group in the comment section when submitting your files
3. Submit the following files to the dropbox (**The files MUST be named exactly like this**):
    1. hw8_p1to5.pdf (Answers for problems 1-5 in PDF format)
    2. hw8_p6.asm (Assembly code for problem 6)
    3. hw8_p7.asm (Assembly code for problem 7)

You may submit your archive file as many times as you want until 9:55 AM on Monday, May 7th. After that time we will consider your latest submissions for grading.

Problem 1 (2 points)
Given the following LC-3 program:

```
          .ORIG x3000
          LD R1, ADDR
          AND R0, R0, #0
          LD R2, NUM
LOOP      LDR R3, R1, #0
          ADD R0, R0, R3
          ADD R1, R1, #1
          ADD R2, R2, #-1
          BRp LOOP
          STI R0, VALUE
          HALT
ADDR      .FILL x4000
VALUE     .FILL x5000
NUM       .FILL xA
```

What does the above program do? Answer using one or two sentences.

Problem 2 (2 points)
An LC-3 program is provided below

```
          .ORIG x3000
          LD R0, ASCII
          LD R1, NEG
AGAIN     LDI R2, DSR
          BRzp AGAIN
          STI RO, DDR
          ADD R0, R0, #2
          ADD R3, R0, R1
          BRn AGAIN
          HALT

ASCII     .FILL x0041
NEG       .FILL xFFB6
DSR       .FILL xFE04   ; Address of DSR
DDR       .FILL xFE06   ; Address of DDR
          .END
```

a. What does this program do?

Problem 3 (3 points)

The following LC3 subroutine checks if the positive integer in R1 is a factor of positive integer in R0. (R0 ÷ R1 has no remainder). If the condition is true R2 is set to 1 else R2 must be set to 0. Fill in the missing parts of code.

```
DIV       AND R2, R2, #0
          ______________
          ADD R1, R1, #1
LOOP      ADD R0, R0, R1
          ______________
          BRn DRET
          ______________
DRET      RET
```

Problem 4 (4 points)

```
          .ORIG X3000
          LEA R2, C
          LDR R1, R2, #0
          LDI R6, C
          LDR R5, R1, #-3
          ST R5, C
          LDR R5, R1, #-4
          LDR R0, R2, #1
          JSRR R5
          AND R3, R3, #0
          ADD R3, R3, #7
          LEA R4, B
A         STR R4, R1, #0
          ADD R4, R4, #2
          ADD R1, R1, #1
          ADD R3, R3, #-1
          BRP A
          HALT
B         ADD R2, R2, #1
          LDR R0, R2, #0
          JSRR R5
          TRAP X29
          ADD R2, R2, #15
          ADD R0, R2, #3
          LD R5, C
          TRAP X2B
          ADD R2, R2, #5
          LDR R0, R2, #0
          JSRR R5
          TRAP X27
          JSRR R5
          JSRR R6
C         .FILL X25
          .STRINGZ "CS252 and tests are awesome"
          .END
```

What does the above program do? Explain briefly.

Problem 5 (4 points)

Before the following program is executed, there is a non-zero value at location IN.

```
                .ORIG x3000
                LD R0, INPUT
                AND R3, R3, #0
                LEA R6, MASKS
                LD R1, COUNT
LOOP            LDR R2, R6, #0
                ADD R3, R3, R3
                AND R5, R0, R2
                BRz SKIP
                ADD R3, R3, #1
                ADD R0, R5, #0
SKIP            ADD R6, R6, #1
                ADD R1, R1, #-1
                BRp LOOP
                ST R3, RESULT
                HALT
COUNT           .FILL #4
MASKS           .FILL 0xFF00
                .FILL 0xF0F0
                .FILL 0xCCCC
                .FILL 0xAAAA
INPUT           .BLKW 1
RESULT          .BLKW 1
                .END
```

What does the above program do? Answer using one or two sentences.

HINT: Check the bit pattern of RESULT for different values of INPUT.

Problem 6 (5 points)

Write an LC-3 assembly program that prompts the user by displaying **"Enter a number from 1 to 9:",** to enter a number between 1 to 9. If the number entered is **3**, the program prints "**DONE**" and terminates. Else it continues to prompt the user to enter a number between 1 to 9 as Input by displaying the string **"Enter a number from 1 to 9:"**.

Problem 7 (20 points)

For this problem you will implement parts of the game "Connect-the-Dots." The game board is comprised of 8 dots in the x-direction and 8 dots in the y-direction. Each player takes turns connecting a dot horizontally or vertically to one of its neighboring dots. If a player completes a 1x1 square by connecting two dots, then that same player may make another move. The objective of the game is to have the most number of completed squares at the end of the game.

Assume that each dot is identified by its **board coordinate**, where [0,0] is the upper left dot and [7,7] is the lower right dot. On the video display, dot [0,0] exists at the **display coordinate** (36, 34) (Note the different notations for coordinates!!!). The value in PLAYER determines who is the current player (1 or 2). Player 1 is RED and Player 2 is BLUE.

Download the template for this problem: hw8_p7.asm. If the format is messed up when you open it, copy and paste the PDF of the code into your desired text editor. If the machine you work on is Mac, TextMate will open it neatly, if you are working on Windows/Linux, the Notepad++ will open it neatly. The template code provided is tested and assembled in Pennsim, so make sure to assemble it with Pennsim if you get assembly error in LC3edit.

DO NOT CHANGE THE LABEL NAMES IN THE SUBROUTINES.
DO NOT CHANGE THE PIECES OF CODE THAT ARE SPECIFICALLY MENTIONED NOT TO CHANGE THEM.
ANY CHANGE IN LABEL NAMES OR THE PIECES OF CODE YOU ARE NOT SUPPOSED TO CHANGE MAY CAUSE THE AUTOMATIC TESTS FAIL AND YOU GET ZERO.

Modify the template according to the tasks below.

1. (3 points) Implement the CHECKFORMAT subroutine in LC-3 assembly code. This subroutine will take in two sets of coordinates for dots that the user intends to connect and output whether it is valid. Assume the following:
    a. The input coordinates are stored in X1, Y1, X2, Y2, where [X1, Y1] is the board coordinate of one dot on the board and [X2, Y2] is the board coordinate of another dot
    b. The output is stored in R0. 0 indicates no error, -1 indicates an error
    c. Rules:
        i. Input coordinates must be in the range [0-7]
        ii. The input coordinates must be a neighbor (1 away) of the other in either the vertical or horizontal direction
        iii. They are invalid if an edge between the two dots already exists. **Hint** CHECKEDGE
2. (2 points) Implement the FINDADDR subroutine in LC-3 assembly code. This subroutine will take a board coordinate [X1, Y1] as an input and calculate the corresponding address
    a. The input coordinate is stored in X1, Y1, where [X1,Y1] is the board coordinate
    b. The output address is stored in R0. **See background information for how to calculate this.**
3. (3 points) Implement the CONNECT subroutine in LC-3 assembly code. This subroutine will take two sets of coordinates for dots to connect and draw a line between them. Assume the following:
    a. The input coordinates are stored in X1, Y1, X2, Y2, where [X1, Y1] is the board coordinate of one dot and [X2, Y2] is the board coordinate of another dot
    b. The color that you must use may be determined by the value in CURRPLAYER
    c. Do not color over the dots

d. Assume that inputs are always valid.

1. (6 points) Implement the CHECK_SQUARE subroutine in LC-2 assembly code. This subroutine will take two sets of coordinates for two dots connected and checks if a square is being made. Assume the following:
   a. The input coordinates are stored in X1, Y1, X2, Y2, where [X1, Y1] is the board coordinate of one dot and [X2, Y2] is the board coordinate of another dot.
   b. Assume that inputs are always valid.
   c. If a square is being made, then you have to call the COLOR_SQUARE subroutine to color it.
   d. Depending on which player wins the square, update the PLAYERONE or PLAYERTWO counter. These counters keeps track of the numbers of squares each player has won.
1. (4 points) Implement the COLOR_SQUARE subroutine in LC-3 assembly code. This subroutine will take two sets of coordinates for dots and will color the area described by the input coordinates.
   a. The input coordinates are stored in X1, Y1, X2, Y2, where [X1, Y1] is the board coordinate of one dot and [X2, Y2] is the coordinate of another dot
   b. Assume [X1, Y1], [X2, Y2] are two dots of the same square that are diagonal from eachother (e.g. [0,0] and [1,1], [3,3] and [4,2])
   c. The color that you must use may be determined by the value in PLAYER
   d. Do not color over the dots
   e. Assume that inputs are always valid
2. (2 points) Implement the CHECK_END subroutines. This subroutine checks if all the dots are connected, or in other words, checks when the game is over.
   a. If all the dots are connected, this subroutine calls the subroutine WHOWON which writes on the console who is the winner and the program stops execution.
   b. Otherwise it returns.

Important Notes:

1. Your files would be tested automatically, so make sure to keep the name of labels and subroutines provided untouched.

2. You can add new subroutines and new variables as much as you need.

3. The file you submit should be named hw8_p7.asm.

4. Not respecting the above rules will result in zero.

Background Information:

Video console interface: The video console is directly linked to a region of addressable memory, namely xC000 to xFDFF. Every row is 128 dots (also called pixels) across, and there are 124 rows in total, which gives us a 128x124 screen to work with. Each pixel is linked to one memory address, so xC000 is the pixel on the top left corner, xC001 is the pixel to the right of it, and so on. Since 128 is x0080 in hex, xC000 + x0080 = xC080 is the location of the leftmost pixel on the second row from the top, and xC000 + 2*x0080 = xC100 would be the next pixel down. In general, you can use the following formula to calculate which address of a pixel:

address = 0xC000 + (y_coordinate * 128) + x_coordinate

Coordinate systems: This code uses just the video console coordinates. This system is zero based (that is, the top-most row is row 0 and the left-most column is column 0).

Color encoding: For the region in memory associated with the video console, each address holds a 16-bit value which determines the color of that pixel. The high-order bit is ignored, and the remaining 15 bits specify the color, using a 5-bit values for each of red [14:10], green [9:5], and blue [4:0]. The codes of the colors you would need for your program (BLACK, WHITE, RED, BLUE) is given to you in the code provided; however if you are curious to know the value for some other common colors, here are a few of them:

Color	**Color Code**
White	0x7FFF
Black	0x0000
Red	0x7C00
Green	0x03E0
Blue	0x001F
Puce	0x3466