Mathematics 3670: Laboratory 10

Background

The LC-3 computer that we have been using in lab is supplied with a **memory-mapped** video display. In the simulator, this display appears under the registers as a nearly square region.

The display is composed of a two-dimensional array of pixels: 124 rows of 128 pixels each. If the rows are numbered from 0–123 and the columns are numbered from 0–127, then any one pixel can be addressed by giving an ordered pair (r, c), where r is the row number and c is the column number. Following usual conventions, the pixel $(0, 0)$ is at the upper left-hand corner. Increasing rows go down; increasing columns go from left to right.

Any one pixel can be described with a 15-bit value, using a simplified RGB color model. Briefly stated, the colors red, green and blue are blended to form a desired color. Each of the three colors is specified with an unsigned 5-bit integer, which specifies the amount of each color. So, for example, red is specified with the triplet $(\texttt{11111}, \texttt{00000}, \texttt{00000})$ — or `x7C00`. Similarly, black is $(\texttt{00000}, \texttt{00000}, \texttt{00000}) =$ `x0000` and white is $(\texttt{11111}, \texttt{11111}, \texttt{11111}) =$ `x7FFF`.

Each pixel in the display is associated with one word of memory in the LC-3 computer. (The most significant bit is unused, followed by five bits each for the colors red, green, and blue.) The pixel at position $(0, 0)$ is associated with the memory address `xC000`; $(0, 1)$ is at `xC001`, $(0, 2)$ is at `xC002` and so on. The video memory is arranged in **row-major** fashion: it is as if the rows of the video display are cut apart and placed end to end in memory.

Pre-lab Exercises

1. How many pixels are in the video display? Give your answer as a hexadecimal value.

   ____

2. Determine the hexadecimal addresses of each of the following pixels:

the pixel at the upper-left corner	`xC000`
the pixel at the upper-right corner	
the pixel at the lower-left corner	
the pixel at the lower-right corner	

3. Suppose you want to light the pixel at row r and column c. Determine a formula for the address of this pixel.

   ____

4. Suppose we want to light four 2×2 squares in various colors, one in each corner of the display. Determine the addresses and values to produce this effect. Place your answers in the table given in Figure 1.

5. Refer to Appendix A of of textbook. What, exactly, do the instructions `JSR` and `RET` do?

6. Carefully review the code given in `lab10a.asm` and see if you agree that the comments accurately reflect the machine instructions.

7. Read the entire lab to see what you are being asked to do for this week's lab.

	Address	Value
upper left (red)	`xC000`	`x7C00`
upper right (green)		
lower left (blue)		
lower right (white)		

Figure 1: Planning to light four 2×2 squares of pixels.

Lab Exercises

1. Obtain copies of this week's lab files and place them in your **lc3** folder.
2. Launch the LC-3 simulator. Scroll to the addresses you determined in Pre-lab exercise 2, and replace the values (currently `x0000`) there with `7C00`. Verify that the red pixels appear in the correct locations. Since each pixel is very tiny, this requires careful inspection.
3. Darken these pixels you just lit by restoring their original values. (Again, just by scrolling through memory — no programming is needed.)
4. Using `lab10a.asm` as a guide, write an assembly language program which will color each of the four corner pixels red. Save your program in `lab10b.asm`. Assemble and load your program; verify that it works correctly.
5. Using your work summarized in Figure 1, scroll through memory and change the sixteen words of memory. Verify that four squares appear in the correct colors and locations. No programming is needed for this step.
6. After you have seen the correct pixels, reset them to their original values.
7. An LC-3 assembly language program to produce these four squares is desired. Instead of thinking of a hexadecimal address for each pixel, think of the (row, column) style of indexing. Using `lab10a.asm` as a model, extend the main program so it colors the four squares in the corners. Place your program in `lab10c.asm`.

   Since 16 pixels need to be lit, there will be 16 subroutine calls. To keep your main program compact, use a loop to produce these subroutine calls — the pixel coordinates can be stored in an array. Assemble and test your program to be sure it works correctly.
8. Instead of working at the pixel level, we now want to think in terms of 4×4 squares of pixels. These, too, can be organized into rows and columns and indexed in a similar way. There are now 31 rows and 32 columns and any one block can be indexed by the ordered pair (r, c) where $0 \le r \le 30$ and $0 \le c \le 31$.

   Using the subroutine `DRAW_PIXEL` as a guide, design a subroutine named `DRAW_BLOCK`. Before invoking this routine, the index values r and c are to be placed in registers `R0` and `R1`; a 15-bit color is assumed to be in `R2`. As a result, your subroutine should light the corresponding 4×4 block of pixels with the specified color.

   Design your subroutine so it does all of the work of setting the pixels — when your code is complete, `DRAW_PIXEL` will no longer be needed. Include code in your main program to light four 4×4 squares in the corners of the graphics display. Place your code in `lab10d.asm`.
9. **Pascal's triangle** is a convenient way to organize binomial coefficients. For example, here are the first five rows:

$$\begin{array}{ccccc} 1 \\ 1 & 1 \\ 1 & 2 & 1 \\ 1 & 3 & 3 & 1 \\ 1 & 4 & 6 & 4 & 1 \end{array}$$

   Observe that $P_{i,j} = P_{i-1,j-1} + P_{i-1,j}$ for all of the "interior" entries. A similar array of values—Pascal's triangle, mod 2—simply records whether or not the value is even or odd:

$$\begin{array}{ccccc} 1 \\ 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{array}$$

For this situation, $P_{i,j} = (P_{i-1,j-1} + P_{i-1,j}) \bmod 2$.

Using `lab10d.asm` as a starting point, write an LC-3 assembly language program which will do the following:

(a) Read a value, `N_ROWS`, from memory which gives the desired number of rows of the Pascal triangle.

(b) Compute and display a graphical view of the Pascal mod 2 triangle. For each 1 entry, display a 4×4 red block. For the 0 entries, display a blue block.

Place your program in `lab10e.asm`. Test your program to ensure it is correct.

Submissions

Before submitting your work, make sure your name appears in each of the programs you wrote. Also, ensure that each of your programs is generously commented.

Create a **lab10** folder and place copies of all `.asm` programs you wrote in this folder. Submit the folder by dragging it onto the EIU submission icon.

Appendix

Contents of lab10a.asm

```
                ;;
                ;; Author: Bill Slough
                ;;
                ;; Example of a subroutine in the LC-3 assembly language
                ;;
                ;; Lights one pixel on the video display
                ;;

                .ORIG x3000

                LD      R0,ROW
                LD      R1,COLUMN
                LD      R2,RED
                JSR     DRAW_PIXEL                      ; draw_pixel(2, 8, RED)
STOP            HALT

ROW             .FILL   2                               ; which row?
COLUMN          .FILL   8                               ; which column?
RED             .FILL   x7C00                           ; desired color

ROWS            .FILL   124                             ; Number of rows of the video display
COLS            .FILL   128                             ; Number of columns of the video display


DRAW_PIXEL
                ;; Draw a pixel on the graphics display unit
                ;;      R0 - row
                ;;      R1 - column
                ;;      R2 - color

                ST      R3,SAVE_R3              ; Save the registers modified within this routine
                ST      R4,SAVE_R4

                ADD     R3,R0,#0                ;
                LD      R4,PIXELS_PER_ROW       ;
                MUL     R3,R3,R4                ;
                ADD     R3,R3,R1                ; R3 = PIXELS_PER_ROW * row + column

                LD      R4,VIDEO_ADDR
                ADD     R4,R4,R3                ; R4 = address of desired pixel

                STR     R2,R4,#0                ; Video[row,column] = color

                LD      R3,SAVE_R3              ; Restore the registers and exit
                LD      R4,SAVE_R4
                RET
SAVE_R3         .BLKW   1                       ; Save area for registers
SAVE_R4         .BLKW   1
VIDEO_ADDR      .FILL   xC000                   ; base address of the video display
PIXELS_PER_ROW  .FILL   128                     ; number of pixels per row
                .END
```