Introduction
In this lab, you will learn use the video output features of the LC-3 Simulator to render colors in the “screen”.

This document assumes you have completed Lab #2 and are using one of the Linux lab machines, e.g. in Moore 207. If that is not the case, please notify the instructor before proceeding.

Overview
Start the simulator as before and note the black box towards the bottom left of the simulator window. This is the video output, and you can change the appearance of each pixel by writing values representing colors to appropriate memory addresses (this is known as “memory-mapped video”).

The memory region for video output ranges from addresses xC000 to xFDFF. The video display is 128 by 124 pixels (15,872 pixels total) and the coordinate system starts from (0,0) at the top left corner of the display.

Since each row is 128 pixels long, in order to find the location exactly one row below a given location, use x0080 to it. For example, if you are outputting to pixel (3, 10), whose memory location is xC18A, then one row immediately below it would be xC20A (=xC18A + x0080).

As a general rule, this is the formula to find the memory location associated with a given (row, col):

\[
addr = xC000 + row \times x0080 + col
\]

Each video output memory location represents one pixel, which means that the value it contains must be formatted as a pixel would be (i.e. RGB format):

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RED</td>
<td>GREEN</td>
<td>BLUE</td>
</tr>
</tbody>
</table>

A value like x7FFF in location xC000 would output a white dot at (0,0).

Here are a few common colors:

<table>
<thead>
<tr>
<th>Color</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>white</td>
<td>x7FFF</td>
</tr>
<tr>
<td>black</td>
<td>x0000</td>
</tr>
<tr>
<td>red</td>
<td>x7C00</td>
</tr>
<tr>
<td>green</td>
<td>x03E0</td>
</tr>
<tr>
<td>blue</td>
<td>x001F</td>
</tr>
</tbody>
</table>
**Tutorial**

Create a new LC-3 assembly language program called colors.asm like this:

```
.ORIG x3000

LD R0, BASE ; put the base address in R0
LD R1, RED  ; put the value for red in R1
LD R2, WHITE ; put the value for white in R2

STR R1, R0, #0; write the value for red to the address in R0

LD R3, ROW ; out x80 into R3
ADD R0, R0, R3 ; change the address to the row below
STR R2, R0, #1 ; use the offset to draw one column over

HALT

BASE .FILL xC000
RED .FILL x7C00
WHITE .FILL x7FFF
ROW .FILL x80

.END
```

Now assemble and load your program. Set the PC to x3000 and step through the instructions. After the first STR instruction is executed, you should see a tiny red dot in the upper left corner of the video output (look closely!). After the second STR is executed, you'll see a white dot just below and to the right of the red dot.

**Exercise #1**

As you probably know, a “loop” is a piece of code that executes numerous times. It generally uses some counter to control how many times to execute the loop.

In the example below, the value in R0 acts as the counter. When we first enter the loop, it is 10. After completing the loop, it is decremented down to 9. At the branch instruction, we jump back to the start of the loop as long as the counter (i.e., the value in R0) is positive.

```
AND R0, R0, #0 ; set R0 = 0
ADD R0, R0, #10; now R0 = 10, this is our counter

LOOP first instruction of loop
    second instruction of loop
and so on. . .

ADD R0, R0, #-1 ; decrement the counter
BRp LOOP ; go back if R0 is still positive
;; otherwise, R0 counted down to zero, so continue from here
```
Modify the colors.asm program from the Tutorial so that it uses a “loop” to draw a row of red pixels, instead of just the single one in the upper left-hand corner.

Since a row is 128 pixels long, you should have a loop that has a counter starting at 128 (however, that value is too big to use as an immediate operand; how will you get it into a register?).

Then, in each iteration of the loop, you should write the value x7C00 to the appropriate address (how will you control which address gets written to?), starting with address xC000 and going up to xC07F.

Your next homework assignment will be built on the program you write for this part of today's lab, so make sure you get this working before moving on. You can work with your neighbor if you'd like, of course!

**Exercise #2**
Make a copy of the program you wrote for Exercise #1 (so that you don't overwrite it) and modify so that, instead of drawing a line straight across the first row, make it look as if the red pixel is moving diagonally across the screen.

That is, on each iteration of the loop:
- change the red pixel to black
- draw a red pixel one column to the right and one row below where it was before

If you use the “Continue” button in the simulator, the pixel will probably fly by too fast, so you'll want to step through this one.

**Exercise #3**
Now modify the program from Exercise #2 so that, instead of having a single pixel moving across the screen, you have a group of four pixels (arranged in a square) traveling together.

Also change the program such that the pixels change color each time they move. The four pixels should all be the same color as each other at the end of each loop iteration, and you can use any color changing scheme you like.