ECE 109 Program 2: `counter.asm`

Spring 2023

This programming assignment must be completed individually. Do not share your code with or receive code from any other student. The only people who may see your code are the instructor and the ECE 109 TAs.

Evidence of copying or other unauthorized collaboration will be investigated as a potential academic integrity violation. **The minimum penalty for cheating on a programming assignment is a grade of -100 on the assignment.** If you are tempted to copy because you're running late, or don't know what you're doing, you will be better off missing the assignment and taking a zero. Providing your code to someone is cheating, just as much as copying someone else's work.

DO NOT copy code from the Internet, or use programs found online or in textbooks as a "starting point" for your code. Your job is to design and write this program from scratch, on your own. Evidence of using external code from any source will be investigated as a potential academic integrity violation.

Description:

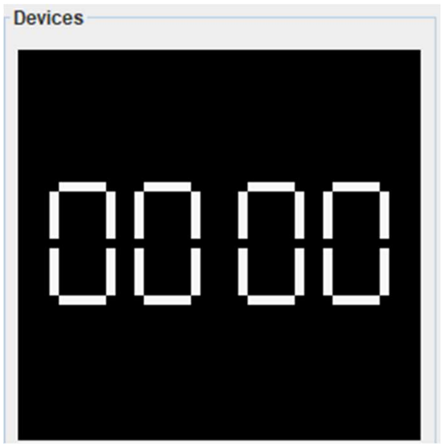
This LC-3 assembly language program will have the user increment and decrement a counter. The counter value has a maximum of 9999 and a minimum of 0000. When the counter reaches 9999 it will wrap around to 0000, and when it reaches 0000 it will wrap around to 9999. The counter value is displayed to the graphics console in 7-segment style digits. The color of the digits is up to the programmer. The counter can be reset to 0000 by the user and the user can print the value of the counter to the console.

The learning objectives for this assignment are:

- Use load and store instructions to manipulate the contents of memory,
- Use I/O routines to allow a user to interact with the program.

Program Specifications:

The program must start at address x3000. The program will start by printing the digits 0000 to the display. Each digit is 25 pixels wide and 40 pixels high. An example of the starting display is shown below:



Each digit is printed to a specific location on the graphics display, the locations are given below:

Digit	Location of top left corner
Thousands	xD508
Hundreds	xD523
Tens	xD544
Ones	xD55F

The color of the digit is what you decide, as long as the digits are easily visible on the display (i.e. do not choose a dark gray or black color). The digit images are provided to you as a separate **digits.obj** file to be loaded into PennSim. The digit image data is stored in row order at the following locations:

Digit	Location in digits.obj
0	x5000
1	x53E8
2	x57D0
3	x5BB8
4	x5FA0
5	x6388
6	x6770
7	x6B58
8	x6F40
9	x7328

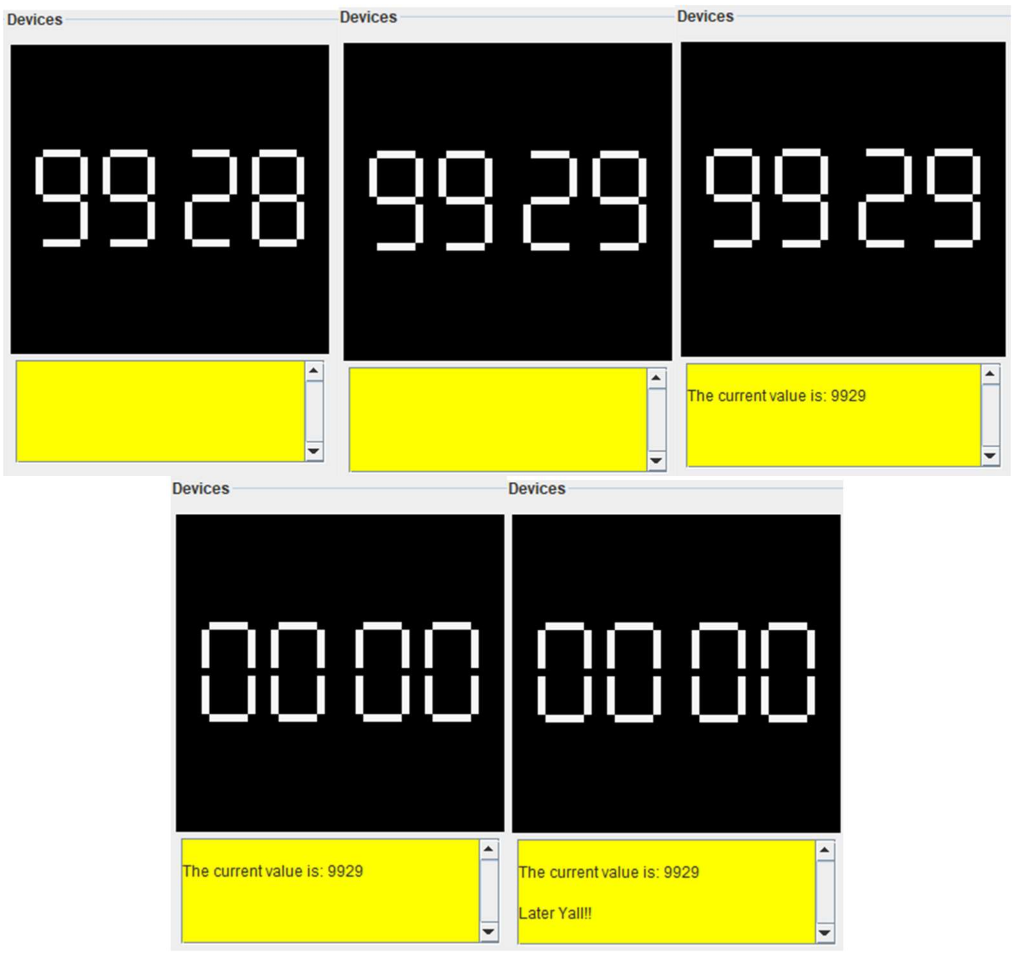
For example, the first row of 25 pixels for digit 0 are stored at x5000 - x5018 and the first row of 25 pixels for digit 6 are stored at x6770 - x6788.

The digits are stored as binary, a 1 for color and a 0 for no color. When drawing the digits, check if that pixel is a 1 or 0. If it is a 1, then fill that pixel with your chosen color, if zero then turn the pixel black. (**Hint:** The color black is the value x0000).

The user interacts with the program using one-character commands. The commands are typed on the keyboard, but are not echoed to the console display. No inputs will be printed to the console during the execution of this program. The program will wait for a keystroke, perform the corresponding command, and repeat. If the keystroke does not correspond to a legal command, it will have no effect. There are five commands for interacting with the counter. The commands and their actions are given below:

Command Character	**Action**
u	*Increment the counter.* The counter value increases by 1 and the graphics display is updated.
d	*Decrement the counter.* The counter value decreases by 1 and the graphics display is updated.
r	*Reset the counter.* Whatever value the counter is, it must reset to 0000 and the display updated.
p	*Print counter value to console.* A newline is printed, then the string "The current value is: ", followed by the value of the counter.
q	*Quit the program.* Print two newlines, then the string "Later Yall!!", followed by one more newline. The simulated machine then must stop running.

Below is an example of the input sequence for 'u' → 'p' → 'r' → 'q', from left to right, up to down.



```
The current value is: 9929
```

```
The current value is: 9929
```

```
The current value is: 9929

Later Yall!!
```

Graphics Display Details:

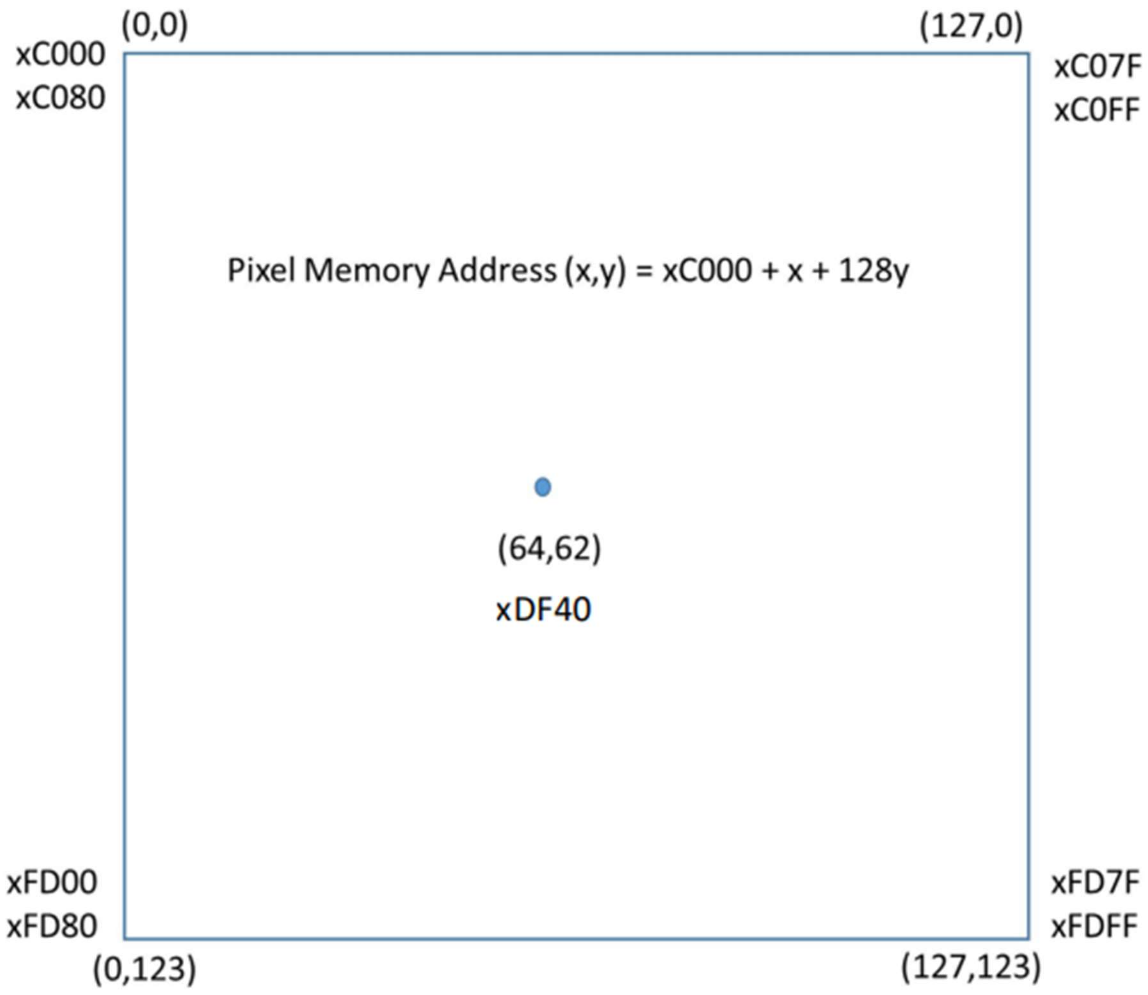
The PennSim graphics display is bit-mapped, meaning that each pixel has a corresponding memory location. The content of that memory location controls the color of the pixel.

Pixel Addresses

Addresses xC000 through xFDFF are assigned to the graphics display. The lower address corresponds to the top left corner (0, 0). Moving one pixel to the right adds one to the address, and it "wraps around" to the next row when it gets to the right edge. Since the display is 128 pixels wide, this means that moving down one pixel is equivalent to adding 128 to the address.

The address of point (x, y) can be calculated as: xC000 + x + 128y.

For this assignment, you will not need to calculate arbitrary pixel addresses, except to figure out where the initial location is. You will be drawing digits that are modeled as rectangles. To draw a rectangle you will need to loop over the width (row) by adding 1 to the address of the point, and then looping over the height (column) by adding a row offset to the address of the point. The offset will be 128 minus the width of the rectangle.



Pixel Color

As mentioned above, the value of the pixel address determines the color of the pixel. The 16 bits contain 5 bits for each RGB component of color: bits [14:10] for red, [9:5] for green, and [4:0] for blue. Bit 15 is ignored. The higher value of a component, the more of that color is present. The table below gives some color values (in hex) that you could use for this program.

Color	**Value**
Red	x7C00
Green	x03E0
Blue	x001F
Yellow	x7FED
White	x7FFF

Miscellaneous

For more explanation about the PennSim display, see the PennSim reference manual.

Hints and Suggestions:

- As always, **design before you code**! Draw a flowchart to organize and document your thinking before you start writing the program. Remember the 80/20 Ratio!!
- Write one section of the program at a time. Then TEST it. Then add the second part. And so on.
- Use PUTS to print a string of characters to the console.
- Use GETC to read a character from the keyboard.
- Since we're using the TRAP instruction (for GETC, etc.), don't put any useful values in R7.
- There are several different ways to solve this problem. Some ways are easier than others, but the important thing is to solve the problem. The TAs may give you some hints, but you should figure out on your own how to solve this problem.
- *Test your program with a wide variety of inputs*. Test corner cases, such as rollover of the thousands digit.
- Use the PennSim simulator and assembler. You must load the file **p3os.obj** into PennSim alongside loading your user code. There are other simulators and assemblers out there, but there are some differences. Your program will be graded using PennSim, and no other tools will be used or considered.
- **Work incrementally!** Implement one thing at a time. When you have a new working piece completed, save and backup a copy of your program. This way you can restore back to a working copy if you need to.
- You are not required to use subroutines but they are recommended to help organize and modularize your code.
- It may be easier to work with the counter value if all digits are stored as ASCII rather than converting between ASCII and binary.
- Recommended order of work:
  1. Figure out how to print digit 0.
  2. Figure out how to print any digit (0-9).
  3. Implement the code that prints all four digits to the graphics display.
  4. Implement your "get input" code that gets user input until 'q' is pressed.
  5. Implement the increment/decrement code that will change the ones digit and update the display when u/d is inputted.
  6. Implement the code that will reset the counter and update the display when r is inputted.
  7. Implement the code that will print the counter value to the console when p is inputted.

8. Update your increment/decrement code to include logic for determining how to increment and decrement the tens, hundreds, and thousands place digits.

Administrative Information:

Updates or clarifications on Moodle:

Any corrections or clarifications to this program spec will be posted on Moodle, using the discussion forum. It is important that you read these postings, so that your program will match the updated specification. (I strongly recommend that you "subscribe" to that forum, so that you get an email for every posting.)

What to turn in:

Assembly Language Source file – it must be named **counter.asm**. Submit via **Moodle**. DO NOT submit .obj or .sym files. Do not submit a .txt file, a .doc file, or anything like that. It must be a simple text file with the proper .asm extension. If we are unable to open or interpret your file, you will get a zero for the assignment (even if it has the right name!).

The program will be graded by one of the TAs, and your grade will be posted in the Moodle gradebook. You will also get back some information about what you got wrong, and how points were deducted (if any).

Grading criteria:

5 pts: Detailed, one full page **flowchart should be submitted to Moodle by the date given!**

5 pts: File submitted with the proper name, **counter.asm**. These are FREE POINTS!

15 pts: **The program is complete.** There is code that makes a reasonable attempt to perform every necessary part of the program's function (print the prompt, read the input, print the result). Assembles with no warnings and no errors using PennSim. Programs that do not assemble will not be graded any further. For warnings, points will be deducted, but the program will be tested for correctness.

10 pts: **Proper coding style, comments, and header.** Use indentation to easily distinguish labels from opcodes. Leave whitespace between sections of code. Use *useful* comments and *meaningful* labels to make your code more readable. Your file **must** include a header (comments) that includes your name, submission date, and a description of the program. Don't cut-and-paste the description from the program spec – that's plagiarism. Describe the program in your own words.

65 pts: The program **performs all functions correctly:**

- (10 pts) Prints all four digits, in the correct locations.
- (10 pts) Input 'p' prints the correct value on the display to the console with the correct string.
- (10 pts) Input 'r' resets the counter value on the display to 0000.
- (15 pts) Input 'u' increments the ones value, 'd' decrements the ones value.
- (15 pts) The tens, hundreds, and thousands digits each rollover if exceeding '0' or '9'.
- (5 pts) Input 'q' halts program after printing the quit string.

<u>Extra Credit:</u>

Extra credit may be earned only if all 100 points have been earned on the base part of the program.

[10 pts] Use the Upper Case keys as shown to change the color of the displayed count numbers.

G:	Green
B:	Blue
R:	Red
Y:	Yellow
W:	White