MP1

MP 1 - Printing histogram

In this first programming assignment (MP), you will extend the code that we started to develop in class to compute a histogram of letters and non-letters in a string. Your final program will print the resulting histogram in hexadecimal to the monitor.

```
Please read the entire document, including the grading rubric, before you begin programming.
```

The Pieces

This week, you are given the histogram code that we started to develop in lecture. You should read through it to make sure that you understand how it works. In Lab 1, we also started to developed a code to print a value stored in a register as a hexadecimal number to the monitor, which involved turning each group of four bits into a digit, calculating the corresponding ASCII character, and printing that character to the monitor. The remaining part of the assignment requires that you use the hexadecimal printing code to print the contents of the histogram to the monitor. For the string shown below, the output produced by a correct program appears below the string.

```
This is a test of the counting frequency code. AbCd...WxYz.

@ 000F
A 0002
B 0001
C 0004
D 0002
E 0005
F 0002
G 0001
H 0002
I 0003
J 0000
K 0000
L 0000
M 0000
N 0003
O 0003
P 0000
Q 0001
R 0001
S 0003
T 0005
U 0002
V 0000
W 0001
X 0001
Y 0002
```

Details

As a first step, we suggest that you study the code provided for this MP to make sure that you understand how it works. You will have to merge the release code into your own repository. For more information on how to do this, take a look at this **Git Setup (https://canvas.illinois.edu/courses/47191/pages/setup)** page.

You can then choose between two pieces. First, you can write the code that manages printing all of the histogram bin labels, spaces, and newlines, and handles the loop control over bins. You will need to write this code yourself at some point. Second, you can start writing code to print a hexadecimal number from a register. We will develop that code as a class during lab 1, but you are welcome to do it yourself ahead of time, if you prefer.

When you are contemplating how to write one of the pieces, start by systematically decomposing the problem to the level of LC-3 instructions. You need not turn in any flow charts, but we strongly advise you not to try to write the code by simply sitting down at the computer and starting.

Once you are ready to start writing code, first write a register table so as to have it in plain sight while you work. You may also want a copy on a piece of paper. Then sketch out the flow of the program using comments. Then write the instructions.

We've added a couple of extra lines of code at the bottom of the .asm file for a simple test (the one given as an example above). When you have debugged a little, the section on testing (on the next page) reminds you of how you can make use of a few scripted tests that we have provided.

Specifics

- Your program must be called `prog1.asm` – we will NOT grade files with any other name
- Your code must begin at memory location `x3000` (just don't change the `.ORIG` code you're given).
- The last instruction executed by your program must be a `HALT` `(TRAP x25)`.
- You must not corrupt the histogram created by the code given to you. You should not change the given part of the code.
- Your output must match the desired format exactly, as shown in this specification and in the test files provided.
- Your program must use an iteration over bins in the histogram.
- You may not make assumptions about the initial contents of any register (before the histogram code executes, that is).
- You may assume that the string is valid.
- You may use any registers, but we recommend that you avoid using `R7`.

Tools

Use a text editor on a Linux machine (vi, emacs, or pico, for example) to write your program for this lab. Use the LC-3 simulator in order to execute and test the program. Your code must work on the EWS lab machines to receive credit, so make sure to test it on one of those machines before handing it in.

Remember to load the LC-3 tools using

`module load lc3tools`

Testing

You should test your program thoroughly before handing in your solution. Remember, when testing your program, you need to set the relevant memory contents appropriately in the simulator. You may want to use a separate ASM file to specify test inputs, as discussed below. When we grade your lab, we will initialize the memory for you. Developing a good testing methodology is essential for being a good programmer. For this assignment, you should run your program multiple times for different functions and inputs and double check the output by hand.

We have also given you three sample test inputs, `testone.asm`, `testtwo.asm`, and `testthree.asm`. You will need to assemble these files. For each test, we have provided a script file to execute your program with the test input `--runtestone`, `runtesttwo`, and `runtestthree` – and a correct version of the output: `testoneout`, `testtwoout`, and `testthreeout`. Look at the script: load the sample input, then load your program. The "reset" command/button will not work, since you are using more than one program. Set the PC by hand if necessary (for example, `r pc 3000`), or re-load both files (input and your program) whenever you need to restart a debug effort.

First you must debug -don't assume that you can simply run the script to debug your code.

Once you think that your code is working, you can execute the script for the first test by typing the following:

```
lc3sim -s runtestone > myoutone
```

This command executes the LC-3 simulator on the script file and saves the output to the file myoutone. If the command does not return, your program is stuck in an infinite loop (press CTRL-C). Otherwise, you can compare your program's output with our program's by typing: diff myoutone testoneout. This will display the lines where your output and the expected output differ.

Note that your final register values need not match those of the output that we provide, but all other outputs must match exactly.

You can also test using the visual LC-3 simulator with

```
lc3sim-tk
```

Keep in mind that you will need to compile and load the test files into the simulator before loading your code.

Handin

Be sure to commit the final version of your program before the deadline for the assignment. Stage any uncommitted changes you have made to the files using `git add ...` to your local repository and then push these changes to the Github repository. For more information on how to do this, take a look at this **Git Setup (https://canvas.illinois.edu/courses/47191/pages/setup)** page.

Grading Rubric

Functionality

- Passes all test cases
- We will provide you a subset of the test cases that will be used to grade with the distributed code in Github
- If your program does not assemble you will get 0.

Style, comments, clarity, and write-up (10%)

- Comments and Style - 5%
- Intro Paragraph - 5%

Note that correct output (and the points awarded for it) also depends on not somehow mangling the contents of the histogram.

Partial credit

We are rewarding partial credit as follows

- If you print 27 bins with incorrect values: 30 pts
- If 1 line is correct (meaning the value is correct but the loop is kind of screwed up): 50 pts
- If most lines are correct but a few have incorrect/malformed values: 60 pts
- -2 points if you use the incorrect new line character (you'll see something like `^M` instead of a new line in their output)

If you fail the tests and don't meet any of the previous criteria you should receive a 0 for functionality.