ECE198KL: Introduction to Computer Engineering II **Spring 2013**
Program 1 **due: 10 p.m. on Monday 21 January**

Printing a Histogram

In this first programming assignment, you will extend code that we 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 Routine

The routine for each week of our class will be the same. You will receive the materials for the week's assignment no later than Monday. Before Thursday, you should at a minimum read through the assignment and the code given to you. We also encourage you to try to do some parts of the assignment before Thursday.

On Thursday, we will meet together in the lab (L440 DCL) for programming studio. We will have a specific piece of the assignment that we work on together as a class. If you have already completed this piece, you can help other people understand how it should be done. If you have run into problems, you will have a chance to ask questions. If you have yet to start the assignment, you will not be able to make much use of the opportunity. **Get in the habit of managing the time that you allot to your assignments; you will need this skill to succeed as an engineer.**

You then have until the following Monday at 10 p.m. to complete the remainder of the assignment, which will make use of the part that we developed in programming studio.

As mentioned in the overview for the course, **all work other than the part developed in programming studio must be your own**. We strongly advise you not to simply take someone else's implementation of the piece that we write together, however. At least sit down with the author and make sure that you understand how their code works.

The Challenges

We plan to try to include "challenges" for those of you who want to learn more about programming. Completion of the challenges **is not required work**, and ignoring them will not affect your grade.

We will assign 20 points to challenges on each assignment. If you are interested in obtaining James Scholar honors credit for this course, your extra work for that distinction will consist of obtaining at least 100 challenge points during the semester. Whether you do everything perfectly on five assignments or do a little bit on all fourteen does not matter, but you need 100 points.

Challenge points can count as extra credit, but please be aware of how I view extra credit. First, extra credit will not affect anyone else's grade: as with last semester, we will have a curve and an absolute scale. I will compute everyone's grades ignoring challenge points. Then I will add points in for those students who have completed some challenges. These points thus neither define nor influence the curve nor the absolute scale, and only affect the grades of those who participate. Second, the work-to-value ratio for challenges will be much higher than the ratio for normal work. Trying to do a challenge in place of the normal programming assignment is a **bad idea**. That balance is deliberate, and the difference will be large. In the end, using my normal scales, completing all of the challenges perfectly may raise your grade by 1/3 of a grade—that's going to be a lot of work for such a small change. Finally, I make no guarantee that a challenge can even be graded if the regular assignment is not working properly, in which case you cannot earn challenge points for your effort.

The Pieces

This week, you are given the histogram code that we developed in class. You should read through it to make sure that you understand how it works.

In programming studio, we will develop code to print a value stored in a register as a hexadecimal number to the monitor, which will involve 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 to the right.

```
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
Z 0001
```

For each of the alphabetic bins, the corresponding capital letter is printed first, followed by a space (ASCII x20), then the four-digit hexadecimal value corresponding to the count for that letter, and finally a newline character (ASCII x0A). The style of the table must match the style of the output to the right exactly to simplify testing. Notice that the non-alphabetic bin is labeled "@," which is the ASCII character before "A," a choice that makes your code somewhat easier to write.

That's it for the required work—all told, including comments and white space, the part after the programming studio took about 40 lines. We've deliberately given you an easy task since you're returning from break and may need some time to remember LC-3. If you'd like, take the opportunity to earn a few challenge points.

Challenges for Program 1

Here are a few challenges if you want more experience writing programs. For this program, they range from fairly trivial extensions requiring a few extra instructions to an extension that requires about the same amount of code but is perhaps more difficult than the main problem (and yet will earn you only 12 challenge points!).

- **(2 points)** Print leading zeroes as spaces instead. (Not printing anything for leading zero digits will not earn full points here.)
- **(3 points)** Allow the user to type the string to be processed using the keyboard (either with GETC or directly using the KBSR and KBDR).
- **(3 points)** Wrap the pieces—finding the histogram, printing the histogram, and printing one hexadecimal number—up as subroutines. You may need to read ahead to do this well.
- **(12 points)** Print the histogram results in decimal rather than in hexadecimal.

Details

As a first step, we suggest that you peruse the copy of the code included with this specification and make sure that you understand how it works.

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 programming studio on Thursday, 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 (the next page explains how to obtain a copy), 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 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.

Checking Out a Copy: A copy of the starting code has been placed in your Subversion repository. To check out a copy, `cd` to the directory in which you want your copy stored, then type:

```
svn co https://subversion.ews.illinois.edu/svn/sp13-ece198kl/<netid>/prog1
```

Replace "`<netid>`" with your Net ID. Recall that the Subversion ("`svn`") checkout command ("`co`") makes a copy of the files stored in your repository in the directory in which you execute the command. The copy will be called "`prog1`"—type `cd prog1` to enter it.

If for some reason we have missed you—possibly if you were not in the roster when we imported the copies to student repositories—you will need to get your own copy and place it into your repository. **Note: if your checkout worked, you can skip these steps.**

1. Check out a copy from the class' shared directory by typing:
   `svn co https://subversion.ews.illinois.edu/svn/sp13-ece198kl/_shared/prog1`
   PDF readers do not copy underscores when cutting and pasting, so you may need to add the one at the start of "`_shared`" back by hand.
2. Change directory into the directory that you have created: `cd prog1`
3. From within the `prog1` directory, import the directory into your subversion account by typing the following **all on one line**, using your own Net ID: `svn import -m "Create program 1 directory." https://subversion.ews.illinois.edu/svn/sp13-ece198kl/<netid>/prog1`
4. Go back out of the `prog1` directory by typing: `cd ..`
5. Remove the copy associated with the class' repository: `rm -rf prog1`
6. Check out a copy of your `prog1` directory as above (before these numerical steps).

Use your working copy of the lab to develop the code. Commit changes as you like, and make sure that you do a final commit once you have gotten everything working. **Commit a working copy and make a note of the version number before you try any of the challenges!**

With your assembly code, you must include a paragraph describing your approach as well as a table of registers and their contents. Avoid using R7 if possible, as any TRAP instructions (such as the OUT traps you will need to print to the monitor) will overwrite its contents and may confuse you.

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 code you're given in that sense).
- 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 that 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.

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`

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

If you implement challenges that change the output, you can find files with which to compare your modified output in the `challenge-outputs` subdirectory.

Handin: Be sure to commit the final version of your program before the deadline for the assignment. (From inside your `prog1` directory, type `svn commit -m "My submission is done."`)

Grading Rubric:

Functionality (50%)
- 15% - program prints one line for each bin
- 15% - program labels each bin correctly in output
- 10% - program prints number of occurrences correctly for each bin
- 10% - program spaces line correctly for each bin

Style (20%)
- 15% - program uses a single iteration to print lines of output
- 5% - program separates code from data (put your new data with the existing data)

Comments, clarity, and write-up (30%)
- 5% - introductory paragraph clearly explaining program's purpose and approach used
- 10% - code includes table of registers **for your new section** that explains their meaning and contents as used by the code
- 15% - code is clear and well-commented (every line)

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

```
;                                                                tab:8
;
; prog1.asm - starting code for ECE198KL Spring 2013 Program 1
;
; "Copyright (c) 2013 by Steven S. Lumetta.
;
; Permission to use, copy, modify, and distribute this software and its
; documentation for any purpose, without fee, and without written agreement is
; hereby granted, provided that the above copyright notice and the following
; two paragraphs appear in all copies of this software.
;
; IN NO EVENT SHALL THE AUTHOR OR THE UNIVERSITY OF ILLINOIS BE LIABLE TO
; ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
; DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION,
; EVEN IF THE AUTHOR AND/OR THE UNIVERSITY OF ILLINOIS HAS BEEN ADVISED
; OF THE POSSIBILITY OF SUCH DAMAGE.
;
; THE AUTHOR AND THE UNIVERSITY OF ILLINOIS SPECIFICALLY DISCLAIM ANY
; WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
; MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE
; PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND NEITHER THE AUTHOR NOR
; THE UNIVERSITY OF ILLINOIS HAS ANY OBLIGATION TO PROVIDE MAINTENANCE,
; SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
;
; Author:           Steve Lumetta
; Version:          1.00
; Creation Date:    11 January 2013
; Filename:         prog1.asm
; History:
;       SL      1.00    11 January 2013
;               Adapted from ECE190 example code.
;



;
; The code given to you here implements the histogram calculation that
; we developed in class.  In programming studio, we will add code that
; prints a number in hexadecimal to the monitor.
;
; Your assignment for this program is to combine these two pieces of
; code to print the histogram to the monitor.
;
; If you finish your program,
;    ** commit a working version to your repository  **
;    ** (and make a note of the repository version)! **
; you may want to try (in increasing order of difficulty)
;   - printing leading zeroes as spaces instead
;   - allowing the user to type the string to be processed using
;     the keyboard (either with GETC or directly using the KBSR
;     and the KBDR)
;   - wrapping the pieces up as subroutines (you may need to read
;     farther ahead to do this well)
;   - printing histogram results in decimal rather than in hexadecimal
;


        .ORIG   x3000           ; starting address is x3000
;
; Count the occurrences of each letter (A to Z) in an ASCII string
; terminated by a NUL character.  Lower case and upper case should
; be counted together, and a count also kept of all non-alphabetic
; characters (not counting the terminal NUL).
;
; The string starts at x4000.
;
; The resulting histogram (which will NOT be initialized in advance)
; should be stored starting at x3F00, with the non-alphabetic count
; at x3F00, and the count for each letter in x3F01 (A) through x3F1A (Z).
;
; table of register use in this part of the code
;    R0 holds a pointer to the histogram (x3F00)
;    R1 holds a pointer to the current position in the string
;       and as the loop count during histogram initialization
;    R2 holds the current character being counted
;       and is also used to point to the histogram entry
;    R3 holds the additive inverse of ASCII '@' (xFFC0)
;    R4 holds the difference between ASCII '@' and 'Z' (xFFE6)
;    R5 holds the difference between ASCII '@' and ''' (xFFE0)
;    R6 is used as a temporary register
;

        LD R0,HIST_ADDR         ; point R0 to the start of the histogram

        ; fill the histogram with zeroes
        AND R6,R6,#0            ; put a zero into R6
        LD R1,NUM_BINS          ; initialize loop count to 27
        ADD R2,R0,#0            ; copy start of histogram into R2

        ; loop to fill histogram starts here
HFLOOP  STR R6,R2,#0            ; write a zero into histogram
        ADD R2,R2,#1            ; point to next histogram entry
        ADD R1,R1,#-1           ; decrement loop count
        BRp HFLOOP              ; continue until loop count reaches zero

        ; initialize R1, R3, R4, and R5 from memory
        LD R3,NEG_AT            ; set R3 to additive inverse of ASCII '@'
        LD R4,AT_MIN_Z          ; set R4 to difference between ASCII '@' and 'Z'
        LD R5,AT_MIN_BQ         ; set R5 to difference between ASCII '@' and '''
        LD R1,STR_START         ; point R1 to start of string

        ; the counting loop starts here
COUNTLOOP
        LDR R2,R1,#0            ; read the next character from the string
        BRz PRINT_HIST          ; found the end of the string

        ADD R2,R2,R3            ; subtract '@' from the character
        BRp AT_LEAST_A          ; branch if > '@', i.e., >= 'A'
NON_ALPHA
        LDR R6,R0,#0            ; load the non-alpha count
        ADD R6,R6,#1            ; add one to it
        STR R6,R0,#0            ; store the new non-alpha count
        BRnzp GET_NEXT          ; branch to end of conditional structure
AT_LEAST_A
        ADD R6,R2,R4            ; compare with 'Z'
        BRp MORE_THAN_Z         ; branch if > 'Z'
```

```
; note that we no longer need the current character
; so we can reuse R2 for the pointer to the correct
; histogram entry for incrementing
ALPHA   ADD R2,R2,R0            ; point to correct histogram entry
        LDR R6,R2,#0            ; load the count
        ADD R6,R6,#1            ; add one to it
        STR R6,R2,#0            ; store the new count
        BRnzp GET_NEXT          ; branch to end of conditional structure

; subtracting as below yields the original character minus '''
MORE_THAN_Z
        ADD R2,R2,R5            ; subtract ''' - '@' from the character
        BRnz NON_ALPHA          ; if <= ''', i.e., < 'a', go increment non-alpha
        ADD R6,R2,R4            ; compare with 'z'
        BRnz ALPHA              ; if <= 'z', go increment alpha count
        BRnzp NON_ALPHA         ; otherwise, go increment non-alpha

GET_NEXT
        ADD R1,R1,#1            ; point to next character in string
        BRnzp COUNTLOOP         ; go to start of counting loop



PRINT_HIST

; you will need to insert your code to print the histogram here



DONE    HALT                    ; done


; the data needed by the program
NUM_BINS        .FILL #27       ; 27 loop iterations
NEG_AT          .FILL xFFC0     ; the additive inverse of ASCII '@'
AT_MIN_Z        .FILL xFFE6     ; the difference between ASCII '@' and 'Z'
AT_MIN_BQ       .FILL xFFE0     ; the difference between ASCII '@' and '''
HIST_ADDR       .FILL x3F00     ; histogram starting address
STR_START       .FILL x4000     ; string starting address

; for testing, you can use the lines below to include the string in this
; program...
; STR_START     .FILL STRING    ; string starting address
; STRING        .STRINGZ "This is a test of the counting frequency code.  AbCd...WxYz."



        ; the directive below tells the assembler that the program is done
        ; (so do not write any code below it!)

        .END
```