ECE199JL: Introduction to Computer Engineering **Fall 2012**
Lab 4 **due: 10 p.m. on Tuesday 11 December**

Printing a Banner

In this laboratory assignment, you will write a short program in LC-3 assembly code to render a string in a large (8x16-pixel) font to the monitor. The objectives for this assignment are for you to gain some experience with assembly language and for you to start to understand how data are organized in memory and how such structured data can be used by a program. In this case, the data of interest map each ASCII character into a picture of the character. By making use of these data to print a string to the monitor, you will begin to learn how to organize data in a way that makes it easy to use in a program.

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

Description: You must write a program in LC-3 assembly language to print a string to the monitor using a large font. We have provided font data that map each extended (8-bit) ASCII character into an 8-bit-wide by 16-bit-high picture of the character. In memory, each such picture is represented using 8 bits in each of 16 contiguous memory locations, and the pictures for each of the 256 possible characters are then simply stored consecutively in memory. The complete set of font data thus occupy a total of $4096 = 16 \times 256$ LC-3 memory locations.

The font data are provided to you in a file called `lab4.asm`. Note that the file includes only a single label, `FONT_DATA`, which points to the start of the data. You must add appropriate assembly code and directives to this file to print a given string to the monitor.

Address	Contents	Meaning
x5000	x0020	space
x5001	x002A	'*'
x5002	x0048	'H'
x5003	x0065	'e'
x5004	x006C	'l'
x5005	x006C	'l'
x5006	x006F	'o'
x5007	x002E	'.'
x5008	x0000	NUL

Input values to your program are stored in memory locations starting at x5000, as shown in the table to the right. Address x5000 contains the character that you should print for 0 bits in the font data. Address x5001 contains the character that you should print for 0 bits in the font data. The string that you must print begins at address x5002 and ends with a NUL (ASCII x00) character. For the example values shown in the table, your program should produce the output shown below. We have added line numbers starting with 0 and colons to both sides of the output to illustrate the inclusion of all spaces.

```
00:                                                  :00
01:                                                  :01
02:**   **          ***     ***                      :02
03:**   **           **      **                      :03
04:**   **           **      **                      :04
05:**   **  *****    **      **    *****             :05
06:******* **   **   **      **   **   **            :06
07:**   ** *******   **      **   **   **            :07
08:**   ** **        **      **   **   **            :08
09:**   ** **        **      **   **   **            :09
10:**   ** **   **   **      **   **   **    **      :10
11:**   **  *****   ****    ****   *****     **      :11
12:                                                  :12
13:                                                  :13
14:                                                  :14
15:                                                  :15
```

The font data for the capital letter 'H' can be seen from the example. The first two memory locations contain x0000. The next four contain xC600 (11000110 followed by eight more 0s). The seventh memory location contains xFE00 (11111110 followed by eight more 0s). The next five memory locations contain xC600. And the last four memory locations contain x0000.

Getting Started: As with Lab #3, your first step should be to systematically decompose the problem to the level of LC-3 instructions. We will not require that you turn in your flow chart, but we strongly advise you not to try to write the code by simply sitting down at the computer and starting.

We suggest that you start by coming up with an algebraic expression for the address that holds the bits needed for a particular line of a particular character. You will find the character to be printed by walking over the string (once for each of the 16 lines to be printed). You need to keep track of the line numbers yourself. You might choose to use one of the LC-3 registers as a loop counter with the line numbers shown in the example, then use the loop counter when calculating the address of the font data for a character. For your convenience, the 8 bits of interest are stored in the upper 8 bits of the word in memory. If we had used the low 8 bits, your code would have to shift each word up after it was loaded in order to use the LC-3 condition code to check the value of each bit. Note that you can use an immediate mode ADD operation with second operand equal to 0 (recall the NOP homework problem) to set the condition codes based on the value in a given register.

Next, think about how you will structure your solution in terms of iterations, conditionals, and sequences. Note that the grading rubric gives a few hints as to what you will need to include, if you're having trouble starting.

When you are ready to start coding, do the following:

1. Log in to an EWS machine.
2. Create a temporary directory, `tmplab4`, and copy the file `lab4.asm` provided to you from the class web site into that directory. This file initially contains only the font data.
3. From within the `tmplab4` 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 lab 4 directory." https://subversion.ews.illinois.edu/svn/fa12-ece199/my_netid/lab4`
4. Now delete the temporary directory (`rm -rf tmplab4` from the parent directory).
5. Check out a copy of your lab4 directory (again, using your Net ID):
   `svn checkout https://subversion.ews.illinois.edu/svn/fa12-ece199/my_netid/lab4`

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.

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.

You may want to start editing by writing a draft copy of the paragraph and register table, then adding some comments that reflect the structure of your program. Finally, add LC-3 instructions to implement each step.

Specifics:

- Your program must be called `lab4.asm` — we will NOT grade files with any other name.
- Your code must begin at memory location x3000.
- The last instruction executed by your program must be a HALT (TRAP x25).
- The character to be printed for 0 bits in the font data is located in memory at address x5000.
- The character to be printed for 1 bits in the font data is located in memory at address x5001.
- The string to be printed starts at x5002 and ends with a NUL (x00) character.
- Use a single line feed (x0A) character to end each line printed to the monitor.
- Your program must use an iteration for each line in the output.
- Your program must use an iteration for each character in the string to be printed. Remember that a string ends with a NUL (x00) character, which should not be printed.
- Your program must use an iteration for each bit to be printed for a given character in the string.
- You may not make assumptions about the initial contents of any register.
- You may assume that the values stored at x5000, x5001, and the string are valid extended ASCII characters (x0000 to x00FF).
- You may use any registers, but we recommend that you avoid using R7.

Note that you must print all leading and trailing characters, even if they are not visible on the monitor (as with spaces). Do not try to optimize your output by eliminating trailing spaces, as you will make your output different from ours (and will thus lose points).

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 a sample test input, `onetest.asm` (you will need to assemble this file), a script file to execute your program with the test input, `runonetest`, and a correct version of the output, `onetestout`. 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 by typing the following:

```
lc3sim -s runonetest > myout
```

This command executes the LC-3 simulator on the script file and saves the output to the file `myout`. 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 myout onetestout`

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

Handin: Be sure to commit the final version of your program using the Subversion commands that you learned in Lab #0 before the deadline for the assignment.

Grading Rubric:

Functionality (50%)
- 30% - program prints string stored starting at x5002 correctly
- 10% - program uses character stored at x5000 to represent 0 bits in font
- 10% - program uses character stored at x5001 to represent 1 bits in font

Style (20%)
- 5% - program uses a single iteration to print lines of output
- 5% - program uses a single iteration to print characters in string
- 5% - program uses a single iteration to print bits in character
- 5% - program uses a single conditional to select character for printing

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