Lab 15

Lab 15 assignment is due on Friday, May 23rd, by 9pm in your svn repository. For help, please check the Labs FAQ first. Specifically, please read the Terminal Troubleshooting and SVN Troubleshooting.

Please ask all questions about this assignment during the office hours or post questions on piazza.

This lab is to be done on **a Linux workstation or** in **VM** during **any** time of the week. Plan your time accordingly since late submissions will not be accepted.

This lab assignment is challenging and time-consuming. Start early, plan your time accordingly. There will be no extensions to submit or correct your code.

If we find your code to be substantially similar to the code of another student, you will receive 0 for the lab and the incident will be reported to the College.

Programming in LC-3 Assembly Language - II

In this laboratory assignment, you will write a program in LC-3 assembly code to render a null-terminated string in a large (8x16-pixel) font to the monitor. You are provided with several files in your SVN repository in the lab15 folder:

- File **lab15.asm** is where you will write your assembly program.
- File **lc-3diff.sh** is a script that compares output files to determine if your code passes/fails the test case.
- File **run_test** is a script file to execute with lc3sim to run your program with a test case. Make sure to modify it to test the different cases provided to you.
- Files **cases/*.asm** contain test cases to help you debug your code. Do not forget to use *lc3as* to assemble each file before the first use.
- Files **cases/*.sol** contain the expected output solution to help you debug your code by comparing to it. They are text files that you can open in your favorite text editor.

Problem Statement

You must write a program in LC-3 assembly language to print a null-terminated 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 × 256 LC-3 memory locations.

The FONT_DATA is arranged according to their ASCII encodings. The first 16 addresses (0-15) in the FONT_DATA correspond to ASCII x00, the NULL character. The next 16 addresses (16-31) correspond to ASCII x01, the SOH character. Use the ASCII encoding of the letters you need to print to similarly find the addresses in FONT_DATA.

The FONT_DATA we provided to you cannot be modified.

The font data are provided to you in a file called lab15.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 lab15.asm to print a given string to the screen that is specified in a separate file. For example, in the file **cases/harry.asm** we have:

Address	Contents	Meaning
x5000	x002E	'.'
x5001	x0040	'@'
x5002	x0048	'H'
x5003	x0061	'a'
x5004	x0072	'r'
x5005	x0072	'r'
x5006	x0079	'y'
x5007	x0021	'!'
x5008	x0020	' ' (space)

x5009	x0000	NULL

Input values to your program are stored in memory locations starting at x5000, as shown in the table above. Address **x5000 contains the character that you should print for every bit set to 0 in the font data**. (In the example above, we will print '.') Address **x5001 contains the character that you should print for every bit set to 1 in the font data**. (In the example above, we will print '@') The **string that you must print begins at address x5002 and ends with a NULL (ASCII x00) character**. For the example values shown in the table above for the file **cases/harry.asm**, your program should produce the output shown below, which you can also find in **cases/harry.sol**

For your convenience, in the figure below we have added line numbers to illustrate the inclusion of all lines. (Line 0 has no number though.)

```
........................................................
........................................................
@@...@@....................................@@...........
@@...@@...................................@@@@..........
@@...@@...................................@@@@..........
@@...@@..@@@@...@@.@@@..@@.@@@..@@...@@...@@@@..........
@@@@@@@.....@@...@@@.@@..@@@.@@.@@...@@....@@...........
@@...@@..@@@@@...@@..@@..@@..@@.@@...@@....@@...........
@@...@@.@@..@@...@@......@@.....@@...@@....@@...........
@@...@@.@@..@@...@@......@@.....@@...@@.................
@@...@@.@@..@@...@@......@@.....@@...@@....@@...........
@@...@@..@@@.@@.@@@@....@@@@.....@@@@@@....@@...........
.....................................@@.................
....................................@@..................
................................@@@@@...................
........................................................
```

Let's focus on understanding the first letter:

```
........
........
@@...@@.
@@...@@.
@@...@@.
@@...@@.
@@@@@@@.
@@...@@.
@@...@@.
@@...@@.
@@...@@.
@@...@@.
........
........
........
........
```

The font data for the capital letter 'H' can be seen from the example. Remember that address x5000 contains the character that you should print for every bit set to 0 in the font data; in the example above, we will print '.' Address x5001 contains the character that you should print for every bit set to 1 in the font data; in the example above, we will print '@' Since we only have to store 8 bits of information, but the LC-3 has an addressability of 16 bits, we have chosen that the 8 bits of interest are stored in the upper 8 bits of the word in memory, and the lower 8 bits are simply set to zero. (We will talk more about this design choice later.) From this, we can observe that first two lines contain x0000. The next four contain xC600 (11000110 followed by eight more 0s). The seventh line contains xFE00 (11111110 followed by eight more 0s). The next five lines contain xC600. And the last four lines contain x0000.

Somewhere in FONT_DATA, you will then find the following information for the capital letter 'H':

```
        .FILL       x0000
        .FILL       x0000
        .FILL       xC600
        .FILL       xC600
        .FILL       xC600
        .FILL       xC600
        .FILL       xFE00
        .FILL       xC600
        .FILL       xC600
        .FILL       xC600
        .FILL       xC600
        .FILL       xC600
        .FILL       x0000
        .FILL       x0000
        .FILL       x0000
        .FILL       x0000
```

By changing the values in memory locations x5000 and x5001 your program should create different visualizations. For example, in the file **cases/hermione.asm** we have:

Address	Contents	Meaning
x5000	x0020	' ' (space)
x5001	x002A	'*'
x5002	x0048	'H'
x5003	x0065	'e'
x5004	x0072	'r'
x5005	x006D	'm'
x5006	x0069	'i'
x5007	x006F	'o'
x5008	x006E	'n'
x5009	x0065	'e'
x500A	x0000	NULL

Your program should produce the output shown below, which you can also find in **cases/hermione.sol**

```


**   **                            **
**   **                            **
**   **
**   **  *****  ** ***  *** **    ***    *****  ** ***   *****
******* **   **  *** ** *******    **   **   **  **  ** **   **
**   ** *******  **  ** ** * **    **   **   **  **  ** *******
**   ** **       **     ** * **    **   **   **  **  ** **
**   ** **       **     ** * **    **   **   **  **  ** **
**   ** **   **  **     ** * **    **   **   **  **  ** **   **
**   **  *****  ****    **   **   ****   *****   **  **  *****




```

Getting Started

Your first step should be to systematically decompose the problem to the level of LC-3 instructions. **We require that you develop your flow chart first**, and we strongly advise you not to try to write the code by simply sitting down at the computer and starting to write without having the flowchart.

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 lower 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 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 iterative constructs, conditional constructs, 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.

An example of the flowchart we expect from you can be found in Patt&Patel's book, page 160, figure **6.3(d)**

Hint: to give you an idea on how to approach this problem, this is how your output should look like if you would run your program step by step and stop it in the middle of the rendering process:

```
........................................................
........................................................
@@...@@....................................@@...........
@@...@@...................................@@@@..........
@@...@@...................................@@@@..........
@@...@@..@@@@...@@.@@@..@@.@@@..@@...@@...@@@@..........
@@@@@@@.....@@...@@@.@@..@@@.@@.@@...@@....@@...........
@@...@@..@@@@@...
```

In other words, you print line-by-line, not character-by-character.

Specifics

Your program must be called lab15.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 NULL (x00) character.
- You may assume that you do not need to test if the string to print is too long, but **do not make any assumptions on the maximum length of the string.**
- Use a single line feed (x0A) character to end each line printed to the monitor.
- Your program must use an iterative construct for each line in the output.
- Your program must use an iteratitive construct for each character in the string to be printed. Remember that **a string ends with a NULL (x00) character, which should not be printed to screen**.
- Your program must use an iteratitive construct 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. That is, **make sure to properly initialize the registers that you will use.**
- 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 (gedit, 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 in the class VM 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, and initialize the contents of registers before you first use them. Developing a good testing methodology is essential for being a good programmer. For this assignment, you should run your program multiple times for different test cases and double check the output with the tools provided to you. Although we provide you with some test cases, you are encouraged to create other ASM files to specify test inputs. **W hen we grade your lab, we will use other test cases to make sure your code works properly.**

We have given you sample test cases in the cases folder, but do not forget to assemble first the ASM files. We have also given you a script file to execute your program with a given case, **run_test**. For your convenience, we have also provided you with the correct version of the output for the test cases in cases/*.sol. The SOL files are text files that you can open in your favorite text editor.

Look at the run_test script: it loads first the sample test, then loads your program. Do not forget to modify this file to run different cases. 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 run_test > myoutput
```

This command executes the LC-3 simulator on the script file and saves the output to the file *myoutput*; 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 solution. For example, assuming you are testing with the **cases/harry.sol** solution, you can type

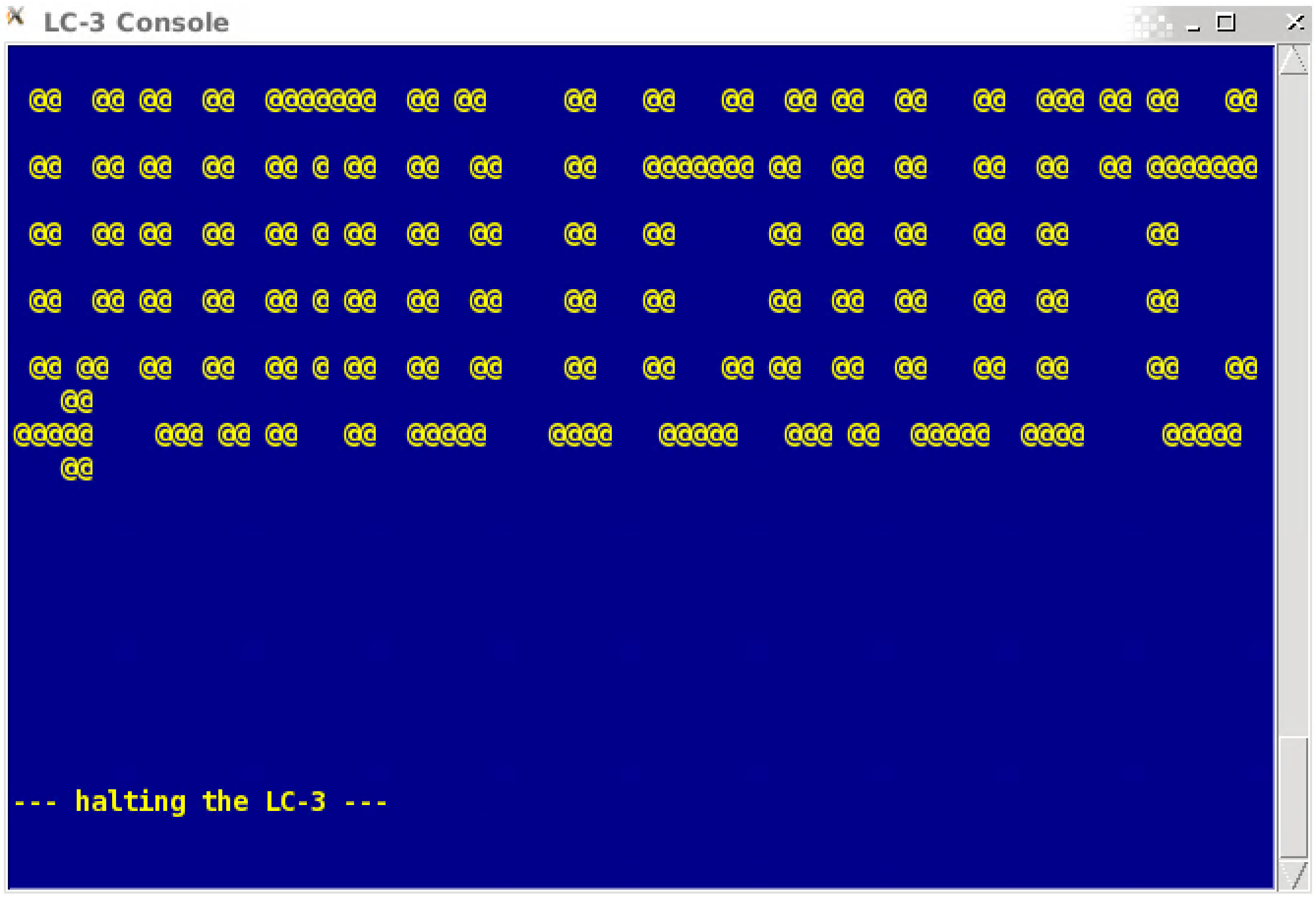
```
./lc3-diff.sh myoutput cases/harry.sol
```

If everything is fine, the message "Test passed" will be printed to screen, otherwise it will be printed "Test failed". The script will give you the opportunity to see your output and our solution so you can manually compare them and see what went wrong.

Do not forget to **run and test your code on the VM machine before submitting** your code, since we will grade your lab using a similar VM.

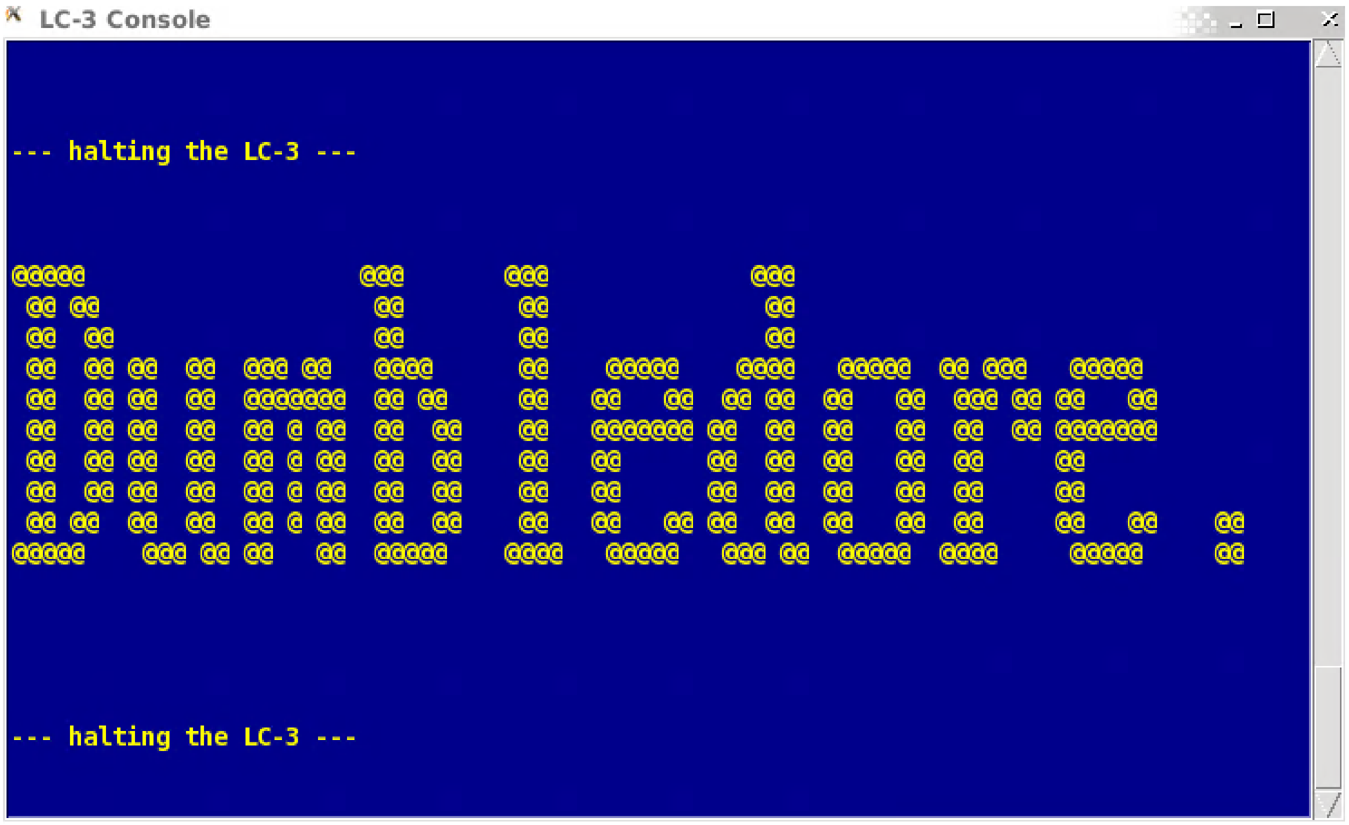
Also, **we will not make any changes to your code after you submitted it**, no matter how small your error is, e.g., you forgot to comment a line that makes your code not to compile at all. It is your responsibility to submit a properly working code.

Hint: if you use the graphical simulator *lc3sim-tk*, you may need to adjust the width of the LC-3 Console when testing long strings. For example, when testing with **cases/dumbledore.asm**, your console could display something like this:



```
 @@  @@ @@  @@  @@@@@@@  @@ @@     @@   @@   @@  @@ @@  @@   @@  @@@ @@ @@   @@

 @@  @@ @@  @@  @@ @ @@  @@  @@     @@   @@@@@@@ @@  @@  @@   @@  @@  @@ @@@@@@@

 @@  @@ @@  @@  @@ @ @@  @@  @@     @@   @@      @@  @@  @@   @@  @@      @@

 @@  @@ @@  @@  @@ @ @@  @@  @@     @@   @@      @@  @@  @@   @@  @@      @@

 @@ @@  @@  @@  @@ @ @@  @@  @@     @@   @@   @@ @@  @@  @@   @@  @@      @@   @@
   @@
@@@@@    @@@ @@ @@   @@  @@@@@    @@@@   @@@@@   @@@ @@  @@@@@  @@@@     @@@@@
   @@








--- halting the LC-3 ---
```

However, after slightly increasing the width of the LC-3 Console, you may get something like this:



```


--- halting the LC-3 ---



@@@@@                   @@@       @@@              @@@
 @@ @@                   @@        @@               @@
 @@  @@                  @@        @@               @@
 @@  @@ @@  @@  @@@ @@   @@@@      @@    @@@@@    @@@@   @@@@@  @@ @@@   @@@@@
 @@  @@ @@  @@  @@@@@@@  @@ @@     @@   @@   @@  @@ @@  @@   @@  @@@ @@ @@   @@
 @@  @@ @@  @@  @@ @ @@  @@  @@    @@   @@@@@@@ @@  @@  @@   @@  @@  @@ @@@@@@@
 @@  @@ @@  @@  @@ @ @@  @@  @@    @@   @@      @@  @@  @@   @@  @@     @@
 @@  @@ @@  @@  @@ @ @@  @@  @@    @@   @@      @@  @@  @@   @@  @@     @@
 @@ @@  @@  @@  @@ @ @@  @@  @@    @@   @@   @@ @@  @@  @@   @@  @@     @@   @@    @@
@@@@@    @@@ @@ @@   @@  @@@@@    @@@@   @@@@@   @@@ @@  @@@@@  @@@@     @@@@@     @@




--- halting the LC-3 ---

```

Lab Submission

Lab 15 submission is to be done via svn before the deadline for the assignment. You should double check that your file (lab15.asm) was uploaded properly.

Grading Rubric

- Functionality (55 points)
    - 50 points - program prints string stored starting at x5002 correctly, using character stored at x5000 to represent bits set to 0 in font and character stored at x5001 to represent bits set to 1 in font, and handles empty string case correctly
    - 5 points - program halts
- Style (35 points)
    - 10 points - program uses a single iterative construct to print every line of output
    - 10 points - program uses a single iterative construct to print every character in string
    - 10 points - program uses a single iterative construct to print every bit in a character
    - 5 points - program uses a single conditional construct to select character for printing
- Comments, clarity, and write-up (15 points)
    - 5 points - introductory paragraph clearly explaining program's purpose and approach used
    - 5 points - code includes table of registers that explains their meaning and contents as used by the code
    - 5 points - code is clear and well-commented