CS 61 - Programming Assignment 03

Objective

The purpose of this assignment is to give you more practice with I/O, and with left-shifting, multiplying by 2, and useful 2's complement logic.

High Level Description

Store a number to the **memory address specified in your assn 3 template**. In your program, load that number in a register, and display it to the console as 16-bit two's complement binary (i.e. display the actual binary value stored in the register)
Note: Valid numbers are [#-32768, #32767] (decimal) or [x0000, xFFFF] (hex)

Your Tasks

You do not yet know how to take a multi-digit decimal number from user input and convert it to binary, so for this assignment you are going to let the assembler to do that part for you: you will use the .FILL pseudo-op to take a literal (decimal or hex, as you wish) and translate it into 16-bit two's complement binary, and store that value in the indicated memory location; and then you will Load that value from memory into R1.

You **MUST** use the provided assn3.asm template to set this up: it ensures that the number to be converted is always stored in the same location (the **memory address specified in your template**) so we can test your work; make sure you fully understand the code fragment we provide.

At this point, your value will be stored in R1: it is now your job to extract the 1's and 0's from the number and print them out to the console one by one, from left *(the leading bit, bit 15, or most significant bit)* to right *(the trailing bit, bit 0, or least significant bit)*.

Important things to consider:

- Recall the difference between a positive number and a negative number in 2's complement binary: if the most significant bit (MSB) is 0, the number is positive; if it is 1, the number is negative.
- The **BR**anch instruction has parameters (n, z, p) which tell it to check whether a value is **n**egative, **z**ero, or **p**ositive (or any combination thereof).
  ***Hint**: what can you say about the msb of the LMR if the n branch is taken?*
  Review the workings of the NZP condition codes and the BR instruction here.
- Once you are done inspecting the MSB, how would you *shift* the next bit into its place so you could perform the next iteration?
  ***Hint**: the answer is in the objectives*

Pseudocode:

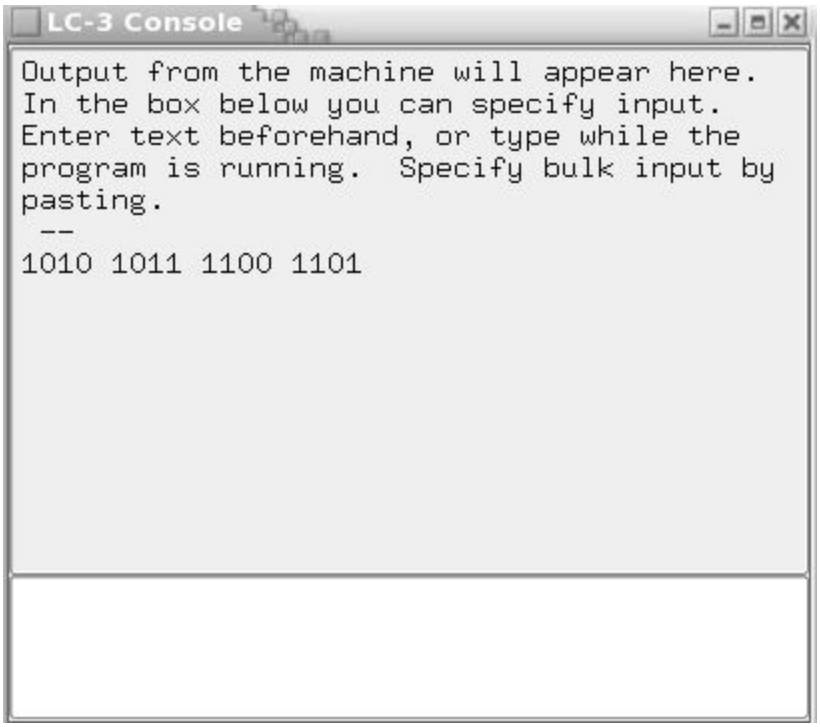
```
for(i = 16 downto 1):
     if (msb is a 1):
          print a 1
     else:
          print a 0
     shift left
```

Expected/ Sample output

In this assignment, your output will simply be the contents of R1, printed out as 16 ascii 1's and 0's, grouped into packets of 4, separated by spaces *(as always, newline terminated, but with NO terminating space!)*

So if the hard coded value was xABCD, your output will be:

```
1010 1011 1100 1101
```



(The value stored to memory with .FILL was xABCD)

Note:

1. *There are **spaces** after every 4 bits, except for the last 4 (i.e. no space character at end)*
2. *There is a **newline** after the output - again, there is **NO** space before the **newline***
3. You **must** use the memory address specified in your template to hold the value to be output

Your code will obviously be tested with a range of different values: Make sure you test your code likewise!

Uh...help?

- **MSB**
  - Stands for Most Significant Bit
    - aka “left most bit” or “leading bit”; in the LC-3, that's bit 15
  - When MSB is 0:
    - Means that the number is **Not Negative** (Positive or Zero)
  - When MSB is 1:
    - Means that the number is **Negative**
  - **Further Reading**
    - https://en.wikipedia.org/wiki/Most_significant_bit

- **Left Shifting**
  Left shifting means that you shift all the bits to the left by 1: so the MSB is lost, and is replaced by the bit on its right. A 0 is "shifted in" to replace the previous LSB.

  Example:

```
0101 ; #5
When Left Shifted:
1010 <---- 0101
1010 ; #10
```

  What happened when we left shifted? How did the number change?
  When left shifting, the number gets multiplied by 2? Why 2?
  Well, what happens when you shift a decimal number one place to the left? Why?
  (Practical differences between decimal and binary numbers are that we don't usually limit decimal numbers to a specific number of places, nor do we usually pad them with leading zeros).

  Further Reading
  - **https://en.wikipedia.org/wiki/Logical_shift**

Submission Instructions

Submit to GitHub *(pull, add, commit, push)*

Comments/Feedback

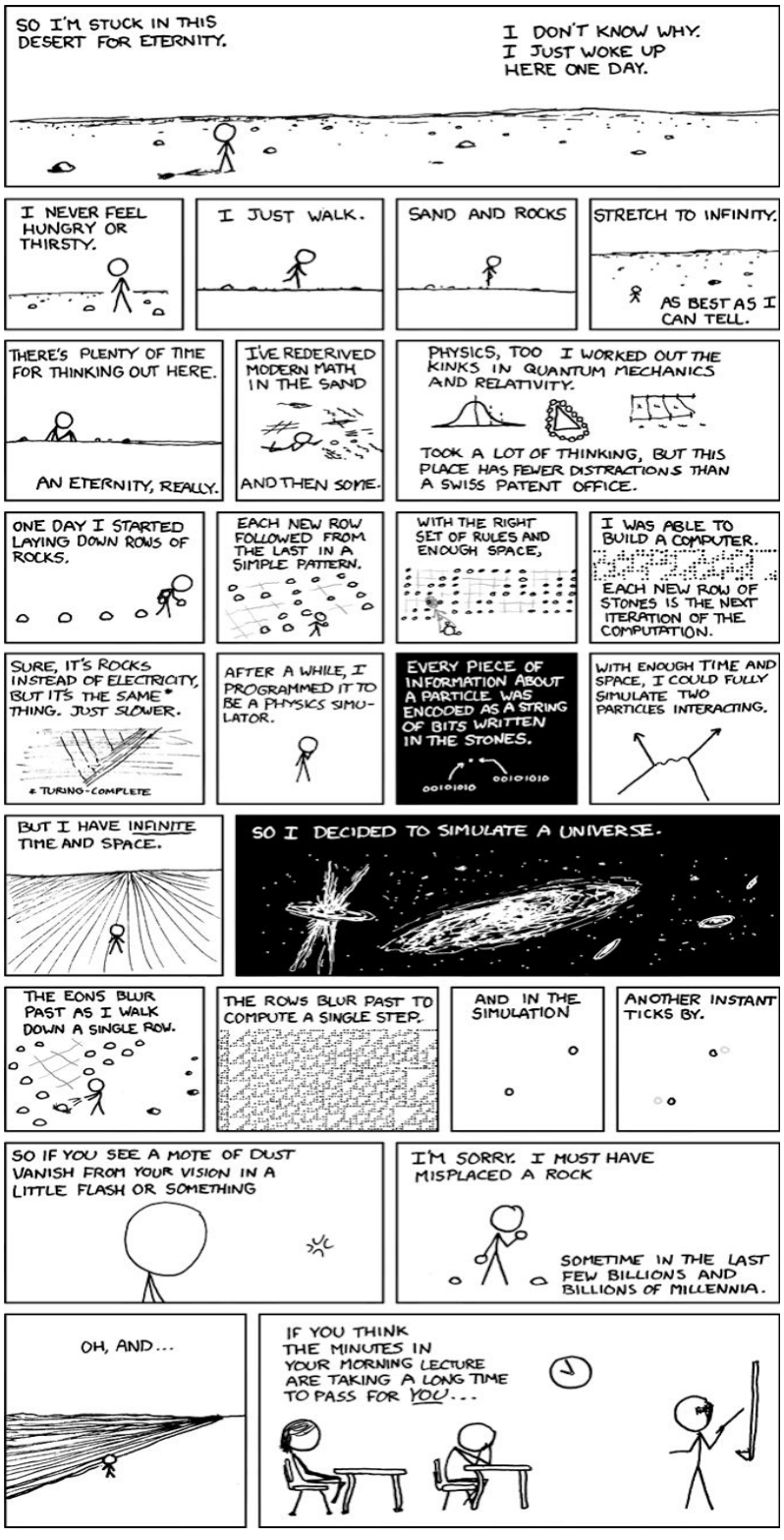
Do a "git pull" to download the results.html file for detailed feedback, every hour on the hour.

Rubric

- The autograder will attempt to assign partial credit for each test (like those described in the expected output section above), and will report which tests passed and which failed.
  To pass the assignment, you need a cumulative score of >= 8/10.
  HOWEVER: after certain errors in your output *(run-time errors, missing newlines, etc. - most of which will be caught & displayed in your Text Window)*, the autograder will be unable to proceed, resulting in a grade of 0/10, with no partial credit.
  This should not be a problem: the problematic output will usually be clearly highlighted in the feedback, so you can fix & resubmit, and hopefully get past the blockage.

- **You must use the template we provide** - if you make *any* changes to the provided starter code, the autograder may not be able to interpret the ouput, resulting in a grade of 0.

Comics??! Sweet!!



Source: http://xkcd.com/505/