CS 61 - Programming Assignment 03

Objective

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

High Level Description

Load any valid number into a register from the **memory address specified in your assn 3 template** and output it to the console as 16-bit two's complement binary (i.e. the native format of the LC-3)
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 get 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 comp. 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 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 to right.

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).
- 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:

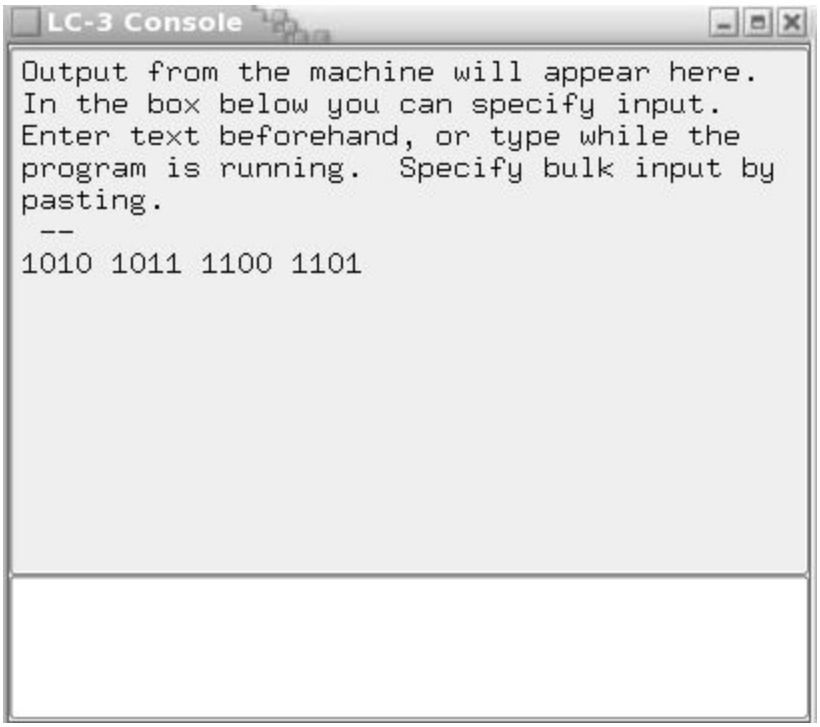
```
for(i = 15 downto 0):
     if (bit[i] is a 0):
          print a 0
     else:
          print a 1
     shift left
```

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.

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

```
1010 1011 1100 1101
```



(Memory address **x7000** contains the value 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 - there is **NO** space before the **newline***
3. You **must** use the memory address specified in your template to hold the value to have program read from

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”
  - 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. When this happens the MSB is lost and is replaced by the bit on its right. The LSB (Least Significant Bit) is replaced by a 0.

  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?
  It is because binary is base 2, so when shifting you are changing the number by a power of 2.

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

Submission Instructions

Submit to GitHub for testing, feedback and grading.

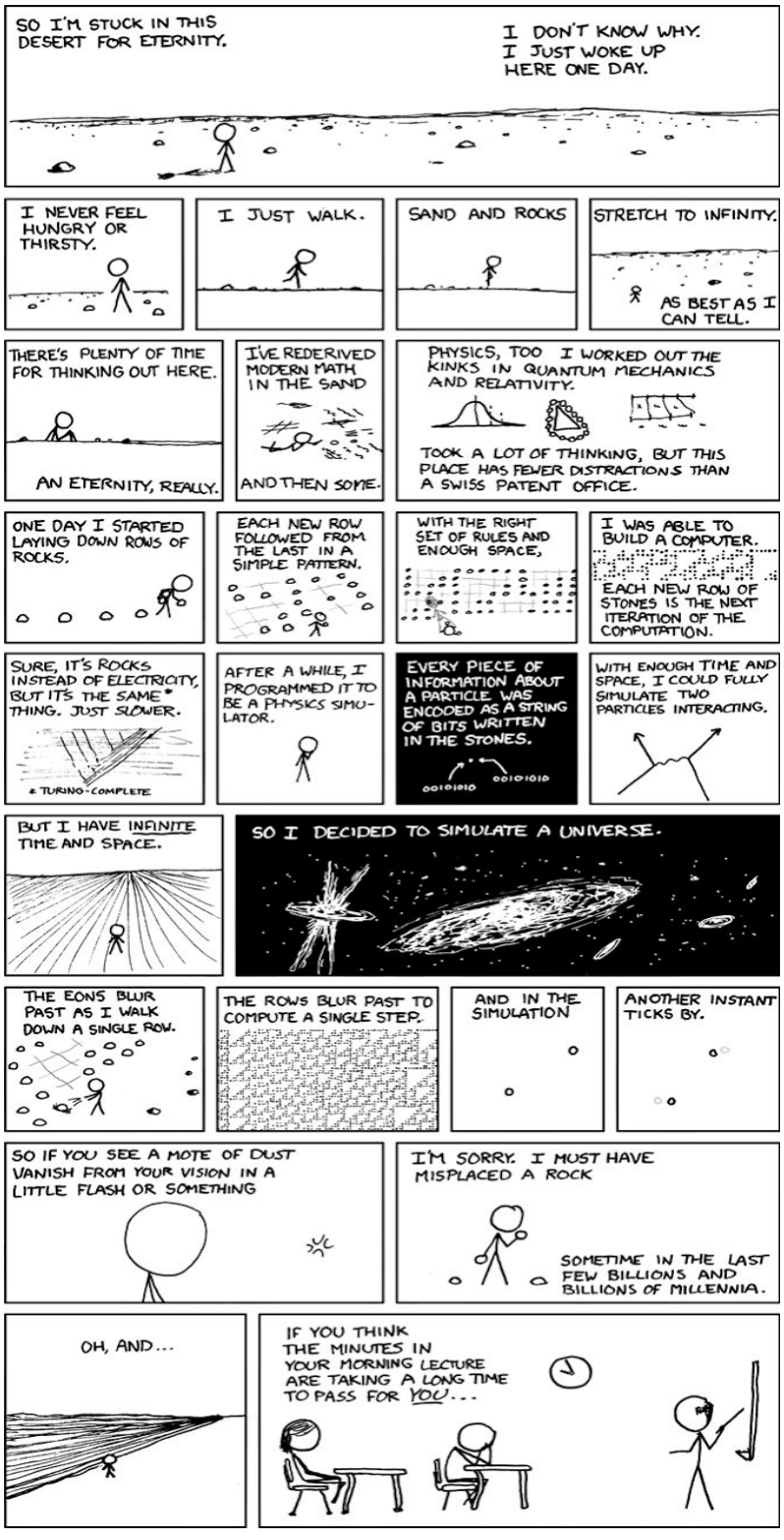
Comments/Feedback

Download the results.html file to see your grade and the reasons for any points deducted.

Rubric

- Grades fall into the following four ranges:
  (i) 8 - 10: pass (perhaps with minor errors - missing spaces, mis-spellings, etc.)
  (ii) 5 - 7: failed, but easily fixable (e.g. missing newlines)
  (iii) 1 - 4: failed - wrong output, you should probably throw it away and start over!
  (iv) 0: abject fail - no submission, or did not assemble *(same as fail, but even more embarrassing!)*
- Using the provided template is **required**.
- Not using memory address provided in the template will result in a failing score.

Comics??! Sweet!!



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