

## CS 61 - Programming Assignment 3

---

### Objective

The purpose of this assignment is to give you more practice with I/O, and with left-shifting, aka 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 to a register, and display it to the console as a 16-bit two's complement binary (i.e. display the binary value stored in the register, as a sequence of 16 ascii '1' and '0' characters).

**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 ascii input and convert it to binary, so for this assignment you are going to let the assembler 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 a 16-bit two's complement binary, which will be stored in the indicated memory location; and then you will Load that value from memory into a register.

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, say, R1: it is now your job to identify the 1's and 0's from the number and print them out to the console one by one, from left (the leading bit, aka the leftmost bit, aka bit 15, aka the most significant bit) to right (the trailing bit, aka the rightmost bit, aka bit 0, aka the 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 considered positive (or zero); if it is 1, the number is negative.
- The **BR**anch instruction has parameters (n, z, p) which tell it to check whether the LMR (Last Modified Register) is **n**egative, **z**ero, or **p**ositive (or any combination of such tests).  
*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 and printing the corresponding ascii '0' or '1', 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 (msb is a 1):
        print a 1
    else:
        print a 0
    shift left
```

### Note on creating LC-3 "control structures"

See [here](#) for tips on creating LC-3 versions of the branch and loop control structures you are familiar with from C++ (Resources -> LC-3 Resources -> LC3 Assembly Language -> Control Structures in LC3)

### 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**



```
Console (click to focus)

1010 1011 1100 1101

--- Halting the LC-3 ---

[]
```

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

### Note:

1. There are **spaces** after the first three "packets" of 4 bits (but 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" or 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](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" on the right to replace the previous LSB.

4-bit Example:

0101 ; #5

When Left Shifted, with 0 shifted in to LSB:

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](https://en.wikipedia.org/wiki/Logical_shift)

## Submission Instructions

Submit ("Upload") your **assignment3.asm** file (*and ONLY that file!*) to the Programming Assignment 3 folder in Gradescope: the Autograder will run & report your grade within a minute or so.

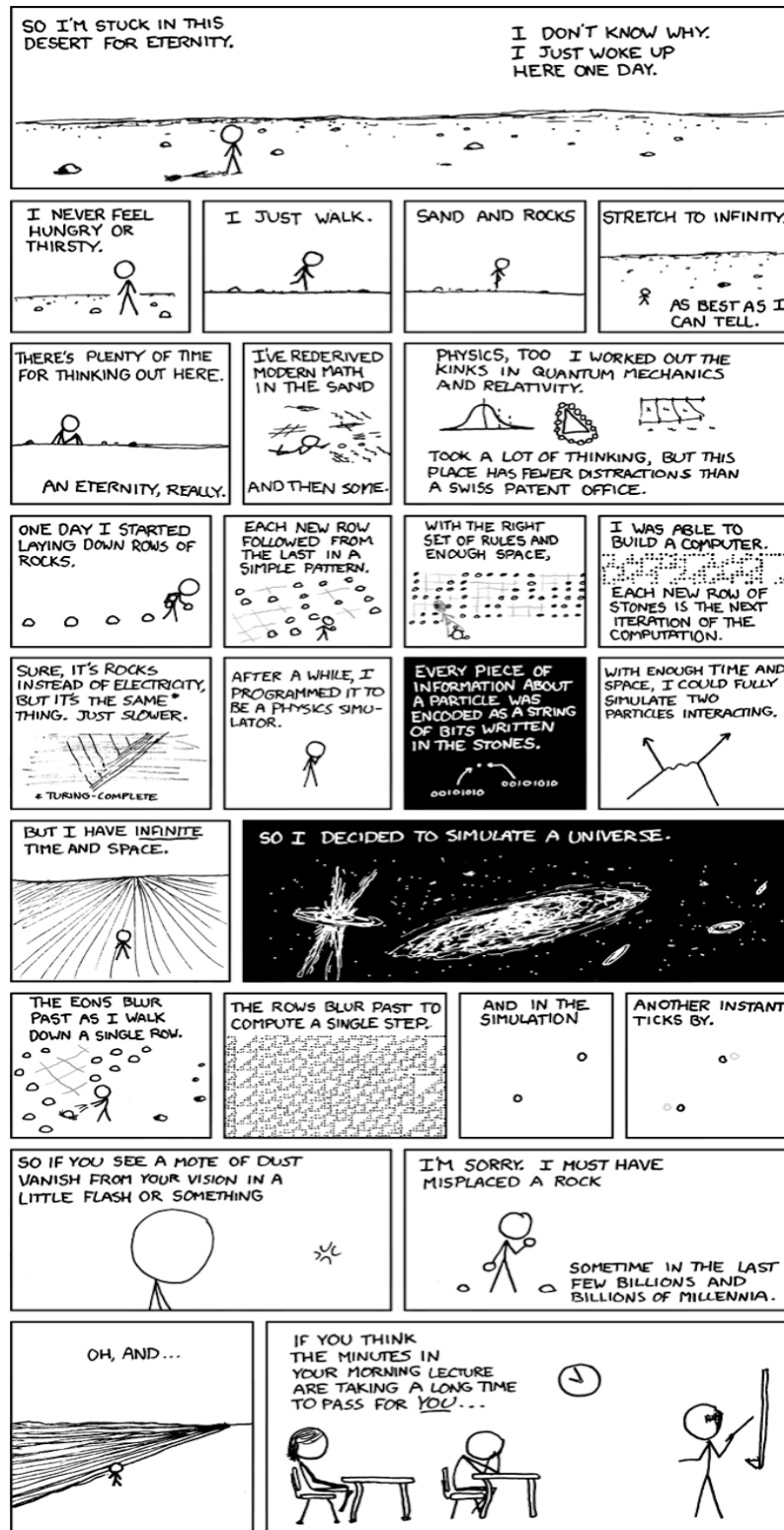
You may submit as many times as you like - your assignment grade will normally be that of your last submission.

*If you wish to set your grade to a previous submission with a higher score, you may open your "Submission history" and "Activate" any other submission - that's the one we will see.*

## Rubric

- To pass the assignment, you need a score of  $\geq 80\%$ .  
The autograder will run several tests on your code, and assign a grade for each.  
But certain errors (*run-time errors, incorrect usage of I/O routines, missing newlines, etc.*) may cause ALL tests to fail  $\Rightarrow 0/100$ ! So submit early and study the autograder report carefully!!
- **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 output, resulting in a grade of 0.

Comics??! Sweet!!



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