

## CS 61 - Programming Assignment 4

---

### Objective

The purpose of this assignment is to illustrate how the .FILL pseudo-op performs the task of translating textual decimal numbers (such as the string "+5392") into actual numbers (i.e. five thousand three hundred and ninety two, represented of course as a 16-bit two's complement binary value).

### High Level Description

Prompt the user to enter a signed multi-digit decimal number (max 5 digits) from the keyboard. Convert the string of decimal numeric digits into the corresponding 16-bit two's complement number, stored in **Rx** (i.e. one of the 8 registers: you will be told which register at the start of the provided starter code).

The range of acceptable values is [-32768, +32767]; the absence of a sign means the number is positive - i.e. the first character entered by the user may be '+', '-', or a numeric digit - and nothing else.

### Your Tasks

Your program can be broken down into the following tasks:

#### 1. Read in the initial character.

- If it is a '-', make the final result negative by setting a "flag" (so if the "negative" flag is set, take the 2's complement of **Rx** at the end).
- If it is '+' it can be ignored;
- If it is a numeric digit, then the number is not negative, and the input is just the first numeric digit of the number.
- In all three cases, you can then proceed to accept any remaining digits (see item 2.)
- If the initial character is a newline, the program can just terminate without any output.
- Any other initial character must be treated as an error (see below).

#### 2. Convert the string of numeric digits input by the user into the binary number they represent (see examples below). To do this, you can follow this algorithm:

- Initialize Rx (and any other registers as needed) to 0 - DO NOT do this by LD'ing 0 from memory! *There is a much simpler & faster way - remember the "TIP" in your lab 1 notes!*
- Convert each digit from ascii code to binary number as it is typed in, and add it to Rx; as each subsequent digit is entered, multiply Rx by #10, and repeat (go back to lab 1 & assn 1 to recall how to multiply!)

Stop when you detect the newline character ('\n' = x0A), **or** when you reach 5 input digits:

- For example, if the user types '2', then Rx will contain  
#2 = b0000 0000 0000 0010
- If the user then types '3', making the string now read "23", then Rx will contain  
 $2 \times \#10 + 3 = \#23 = \text{b0000 0000 0001 0111}$
- If the user then types '4', making the string read "234", then Rx will contain  
 $\#23 \times \#10 + 4 = \#234 = \text{b0000 0000 1110 1010}$

You must also perform **input character validation** with this assignment – i.e. reject any non-numeric input character. That is, if the user enters "+23g", on detecting the non-numeric 'g', your program should output an error, discard all input, and start over with the initial prompt (see sample output).

You must also **count** the number of characters entered - once it gets to 5 you should stop accepting new characters, and issue a newline (in this case, do not wait for the user to hit Enter).

However, you do not have to detect overflow in this assignment – we will only test your code with inputs in the range [-32768, +32767].

### Expected / Sample output

#### Output

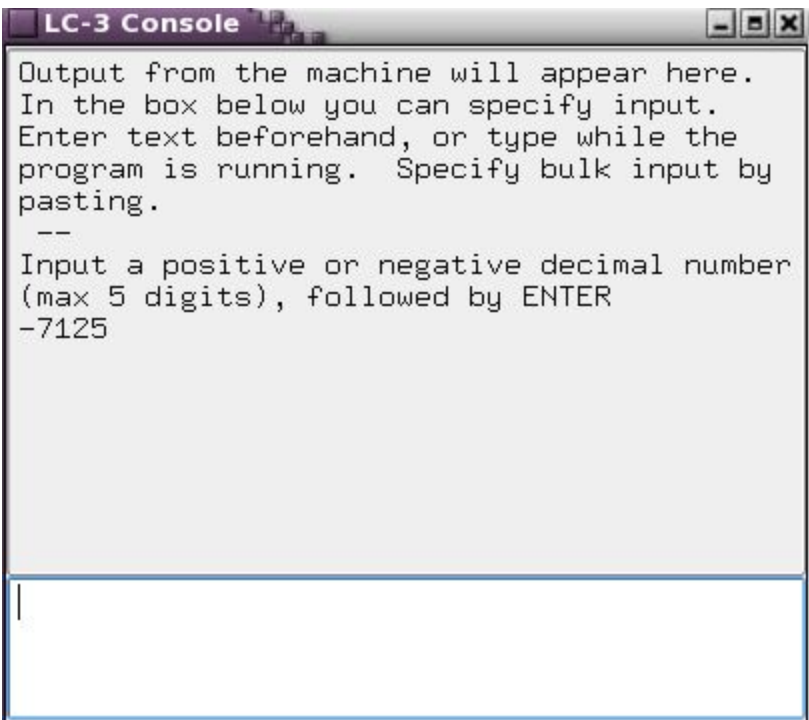
- Prompt
  - "Input a positive or negative decimal number (max 5 digits), followed by ENTER\n"
- Error Message
  - "ERROR! invalid input\n"

#### Example

If the user enters "+7246", your program should read the '+', '7', '2', '4', '6' and end up with the value b0001 1100 0100 1110 (which is the two's complement representation of the number #7246, or x1C4E) in the specified register.

If the user enters "-14237", your program should read the '-', '1', '4', '2', '3', '7' and end up with the value #-14237 = xC863 = b11001000 01100011 in the specified register.

**WARNING:** In the following examples, the final result is shown in R2, which may NOT be the one you will use. You will store your result in the register specified in the header in your starter code!! Make sure you get this right, or the grader will not work, and you will get 0/10!



The screenshot shows the LC-3 Console window. The text inside the console reads: "Output from the machine will appear here. In the box below you can specify input. Enter text beforehand, or type while the program is running. Specify bulk input by pasting. -- Input a positive or negative decimal number (max 5 digits), followed by ENTER -7125". Below the console window, there is a table of register values:

R0	x000A	10	R1	x0000	0	R2	xE42B	-7125	R3	x0004	4
R4	xFFFF6	-10	R5	x0000	0	R6	xFFFF	-1	R7	x3049	MAX_INPUT

(Valid input with a negative sign)

LC-3 Console

Output from the machine will appear here.  
In the box below you can specify input.  
Enter text beforehand, or type while the program is running. Specify bulk input by pasting.

--

Input a positive or negative decimal number (max 5 digits), followed by ENTER

+12345

R0	x000A	10	R1	x0000	0	R2	x3039	12345	R3	x0005	5
R4	xFFF6	-10	R5	x0000	0	R6	x0001	1	R7	x3049	MAX_INPUT

(Valid input with a positive sign)

LC-3 Console

Output from the machine will appear here.  
In the box below you can specify input.  
Enter text beforehand, or type while the program is running. Specify bulk input by pasting.

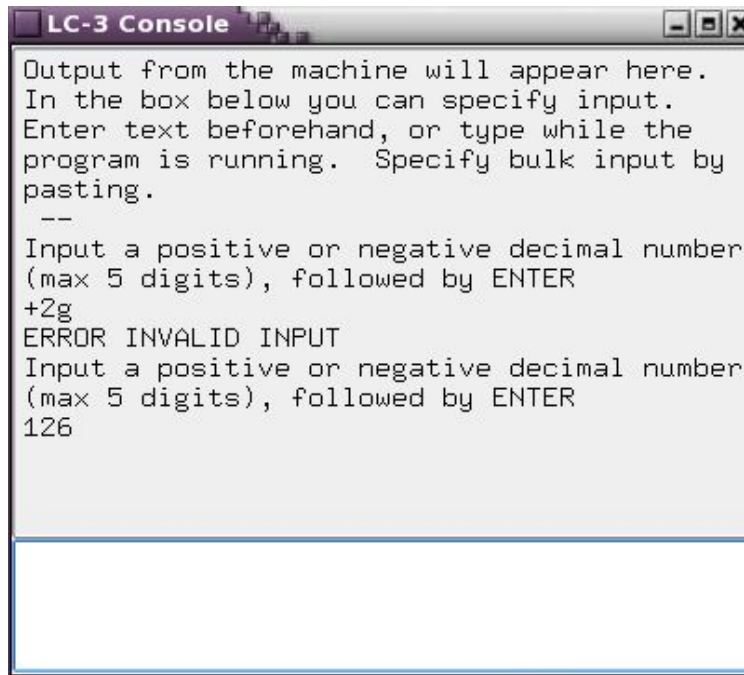
--

Input a positive or negative decimal number (max 5 digits), followed by ENTER

123

R0	x000A	10	R1	x0000	0	R2	x007B	123	R3	x0003	3
R4	xFFF6	-10	R5	x0000	0	R6	x0000	0	R7	x3049	MAX_INPUT

(Valid input with No sign)



R0	x000A	10	R1	x0000	0	R2	x007E	126	R3	x0003	3
R4	xFFFF6	-10	R5	x0000	0	R6	x0000	0	R7	x3049	MAX_INPUT

(Invalid initial input, followed by valid input)

#### Note:

- You must echo the digits as they are input (no "ghost writing").
- You do **not** have to output the converted binary number. It should simply be sitting happily in **Rx**, where you can check it in the simpl interface.
- Remember, the register in which to store your result is given to you in your code header.**
- What should happen when an error occurs?
  - Output "ERROR! invalid input" and start over, re-prompting the user for input
- Possible errors (*we will test for these!*):
  - Nothing entered before ENTER  
(*this does not trigger an error message, it just terminates the program*)
  - only a sign is entered, no numeric digits - *error message, start over*
  - the first character entered is neither a sign nor a numeric digit - *error message, start over*
  - A subsequent character is not a digit - *error message, start over*  
- e.g. the sign character is entered twice, or a letter is entered in place of a digit
- The entire program must end with a single newline.**  
EITHER the user terminates digit entry with Enter ( < 5 digits), OR the program issues its own newline after counting 5 digits - not both!

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

#### Uh...help?

Try writing this program out in C++/pseudocode before tackling it in LC3. Doing so often helps to simplify the process and usually only takes a few minutes if you think it through carefully.

To "flag" a negative number, initialize a designated register (say Ry) to 0, then set it to, say, -1 if the first character entered is a '-'.  
Treat the subsequent numeric digits as a positive number. Once you have translated that number into binary, test Ry and BRz MAKE\_NEGATIVE to take the 2's complement of the result if required.

## Submission Instructions

Submit ("Upload") your **assignment4.asm** file (*and ONLY that file!*) to the Programming Assignment 4 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 grade will 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/237/>