CS 61 - Programming Assignment 04

Objective

The purpose of this assignment is to illustrate how the .FILL pseudo-op performs the task of translating textual 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 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 on the first line 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.

Your Tasks

Your program can be broken down into the following tasks:

Read in the initial character. If it is a '-', remember to make the final result negative by setting a "flag" *(i.e. if the "negative" flag is set, take the 2's complement of **Rx** at the end).*
If the initial character is '+' or a numeric digit, then the number entered is not negative.
Any other initial character must be treated as an error (see below)

Convert the string of characters 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 a 0 from memory! There is a much simpler & faster way!)

- Convert each digit from ascii code to binary number as it is typed in, and add it to **Rx**; as subsequent digits are entered, multiply **Rx** by #10, and repeat. <u>Stop</u> when you detect the newline character (x0A):
  - 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
    $2 \times \#10 + 3 = \#23 = b0000\ 0000\ 0001\ 0111$
  - If the user then types '4', making the string read "234", then **Rx** will contain
    $\#23 \times \#10 + 4 = \#234 = b0000\ 0000\ 1110\ 1010$

You must also perform ***input character validation*** with this assignment – i.e. reject any <u>non-numeric</u> input character. That is, if the user enters "+23g", on detecting the non-numeric 'g', your program should output an error, and <u>start over with the initial prompt</u> (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 (i.e. in this case, <u>do not wait</u> for the user's newline).

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"
    - Newline terminated
- Error Message
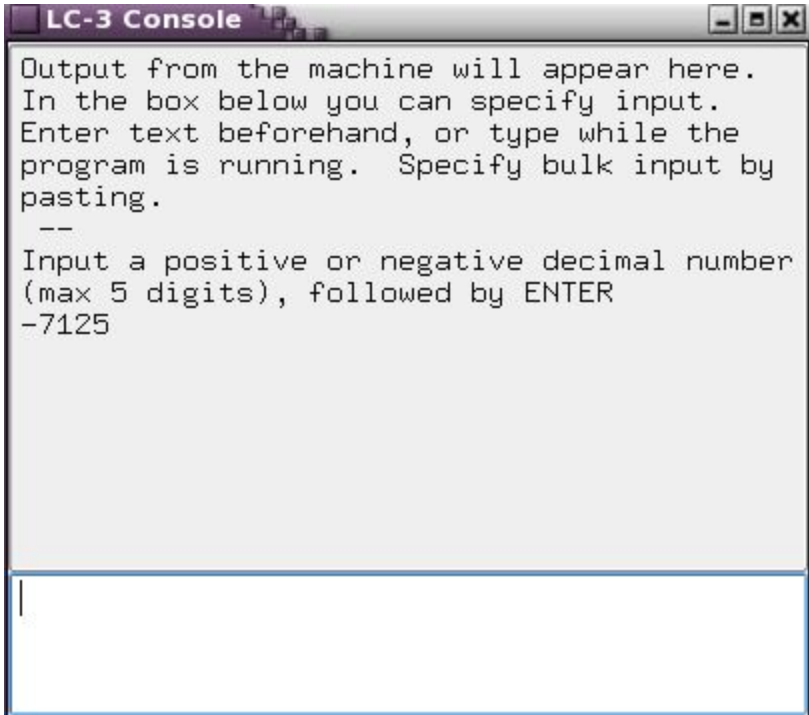 - "ERROR INVALID INPUT"
    - Newline terminated

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 users 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 is NOT what you will use. You will store your result instead in the register specified at the bottom of the header in your starter code!!***
Make sure you get this right, or the grader will not work, and you will get 0/10!

```
 LC-3 Console                        _ □ X
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



 |

```

| R0 x000A | 10 | R1 x0000 | 0 | R2 xE42B | -7125 | R3 x0004 | 4 |
| R4 xFFF6 | -10 | R5 x0000 | 0 | R6 xFFFF | -1 | R7 x3049 | MAX_INPUT |

(Valid input with a negative sign)

```
LC-3 Console                                    - □ X

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




┌──────────────────────────────────────────┐
│                                          │
│                                          │
│                                          │
└──────────────────────────────────────────┘
```

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

(Valid input with a positive sign)

```
LC-3 Console                                    - □ X

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




┌──────────────────────────────────────────┐
│ |                                        │
│                                          │
│                                          │
└──────────────────────────────────────────┘
```

| RO x000A | 10 | R1 x0000 | 0 | R2 x007B | 123 | R3 x0003 | 3 |
| R4 xFFF6 | -10 | R5 x0000 | 0 | R6 x0000 | 0 | RZ x3049 | MAX_INPUT |

(Valid input with No sign)

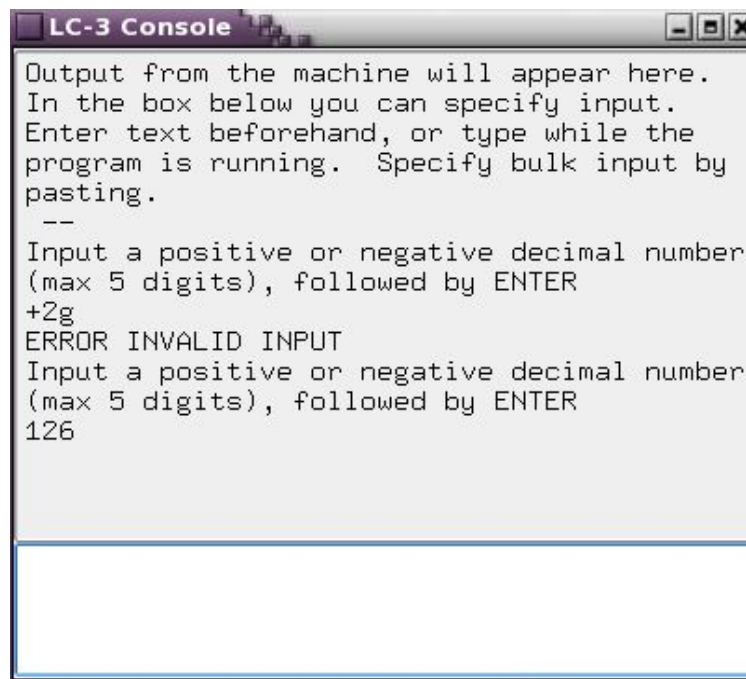```
┌─ LC-3 Console ───────────────────── ─□X
│ Output from the machine will appear here.
│ In the box below you can specify input.
│ Enter text beforehand, or type while the
│ program is running. Specify bulk input by
│ pasting.
│ --
│ Input a positive or negative decimal number
│ (max 5 digits), followed by ENTER
│ +2g
│ ERROR INVALID INPUT
│ Input a positive or negative decimal number
│ (max 5 digits), followed by ENTER
│ 126
│
├──┤
│   │
│   │
│   │
└──┘
```

| R0 x000A | 10  | R1 x0000 | 0   | R2 x007E | 126 | R3 x0003 | 3         |
|----------|-----|----------|-----|----------|-----|----------|-----------|
| R4 xFFF6 | -10 | R5 x0000 | 0   | R6 x0000 | 0   | R7 x3049 | MAX_INPUT |

(Invalid initial 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, prompting the user for input
- Possible errors *(we will test for these!)*:
  - Nothing entered before ENTER
  - only sign is entered, no numeric digits
  - first character entered is neither a sign nor a numeric digit
  - Any subsequent character is not a digit
    - e.g. the sign character is entered twice, or a letter is entered in place of a digit
- ***REMEMBER: all outputs must be newline terminated***

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 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 (ADD Ry, Ry, #0) and BRp MAKE_NEGATIVE to take the 2's complement of the result if required.

**Submission Instructions**

Submit to GitHub for testing, feedback and grading.

**Comments/Feedback**

Do a "Git pull" to download the results.html file for detailed feedback.

**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.
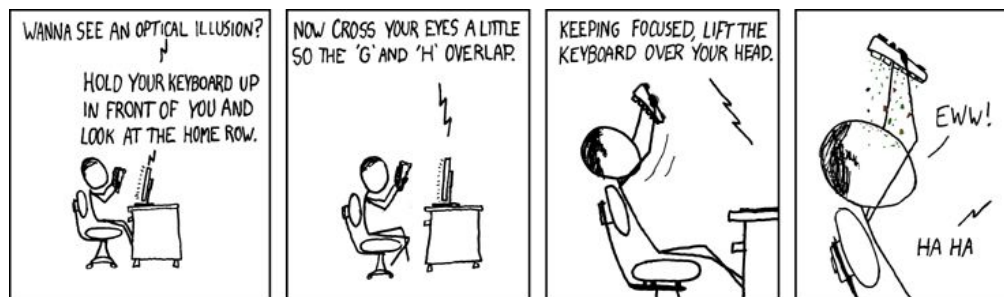  <u>HOWEVER</u>: after certain errors in your output (run-time errors, missing newlines, etc), the autograder will be unable to proceed, resulting in a grade of 0/10, with no partial credit.

  *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.
  *(Unless, of course, you waited until one hour before the deadline to submit, in which case you're stuck with the 0/10 - but you would never do that, would you?)*

- **You must use the template we provide** - if you make <u>*any*</u> 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/237/