CS061 – 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 in a signed multi-digit number (max 5 digits) from the keyboard. Convert the string of characters entered (as separate ascii codes for decimal numeric digits) into the 16-bit number they represent, and store the result in R1. The range of acceptable values is [-32768, +32767]; the absence of a sign means the number is positive.

Examples

If the user enters "+7246", your program should read the '+', '7', '2', '4', '6' and end up with the value b0001 1100 0100 1110 in R1 *(which corresponds to the number #7246, or x1C4E)*.

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 R1.

Your Output

Your program should output a prompt, asking the user to input a positive or negative decimal number and telling the user about any assumptions/requirements. For example:

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

You must echo the digits as they are input *(no ghost writing, please!!)*.
You do **not** have to output the converted binary number. It should "simply" be sitting happily in R1.

Your Tasks

Your program can be broken down into the following tasks:

Read in the '+' or '-'. If the character is a '-', remember to make the final result negative *(i.e. take the 2's complement of R1 at the end).* If the result is positive then ... don't.

Convert the string of characters input by the user into the binary number they represent (see examples). To do this, you can follow one of (at least) two different algorithms:

1. Method 01: Read in the entire string of digits, store them into an array, and then traverse the array backwards while adding each value from the array (weighted by the appropriate power of 10) to the "running total" in R1)..
   So $+1234 = 4 \times 10^0 + 3 \times 10^1 + 2 \times 10^2 + 1 \times 10^3$

2. Method 02 *(same basic algorithm as method 01, but performed "on the fly")*:
   Convert each digit to binary as it is typed in, and add it to R1; if another digit is entered, multiply R1 by 10, and repeat. Stop when you detect the ENTER (x0A):

a. For example, if the user types '2', then R1 will contain
   #2 == b0000 0000 0000 0010
b. If the user then types a '3', making the string now read "23", then R1 will contain 2 x 10 + 3 == #23 == b0000 0000 0001 0111
c. If the user then types '4', making the string read "234", then R1 will contain
   23 x 10 + 4 == #234 == b0000 0000 1110 1010

You must also perform *input character validation* with this assignment – reject any non-numeric input character. That is, if the user enters "+23g", your program should choke on the 'g', print an error message of some kind, and start over at the beginning (not just HALT). However, you do not have to detect overflow – we will only test your code with inputs in the range [-32768, +32767].

Uh…help?

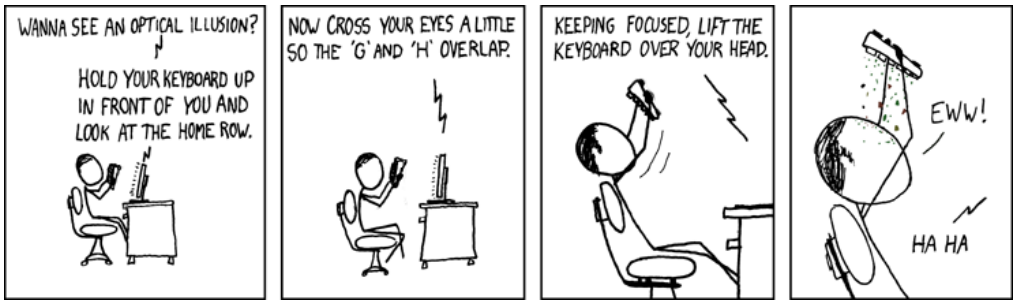
Try to write this program out in C++/pseudocode before directly tackling it in LC3. Doing so often helps to simplify the process and *usually* only takes a few minutes to write out if you think it through carefully.

To mark the distinction between a positive number and a negative one, set a "flag" (say… R5). If the first character is a '-', then put a negative number (like #-1) into R5. Otherwise, set R5 to #0 (i.e. non-negative). That way, after you translate the rest of the input characters into the number they represent, you can use a quick IF-statement (like BRn MAKE_NEGATIVE) to toss in the two lines of code it takes to take the 2's complement of the result.

Rubric

- Code does not assemble: -10 points *(no reshow)*
- No header: -10 points *(no reshow)*
- Well styled and commented code: +2 points
- Works for positive inputs: +3 points
- Works for negative inputs: +3 points
- Input validation& restart on input error: +2 points

Comics?!Sweet!!



Source: http://xkcd.com/237/