CS061 - Programming Assignment 2

Objective

To further familiarize you with the basic LC-3 instructions;
to understand the difference between numeric characters and actual numbers;
to handle two's complement conversions;
and to perform basic input/output.

High Level Description

Prompt the user to input two single digit numbers.
The second will then be subtracted from the first, and the operation reported in the console:

```
<first number> - <second number> = <difference>
```

SO if the user enters 8 and 4, these two numbers will first be echoed to the console on separate lines, then the subtraction operation will be displayed:

```
ENTER two numbers (i.e '0'....'9')
8
4
8 - 4 = 4
```

Note: the user does not enter any newlines; all newlines are to be generated by the program itself.

LC-3 I/O

First, review the ASCII Table, and read the "LC-3 Basic Input/Output" document in Canvas at FIles/General Resources/LC-3 Assembly Language Guides/LC-3 Basic Input_Output.docx

Low Level Breakdown

This assignment comprises five tasks:

1. Prompt the user, and read two numeric characters *('0' ... '9')* from the user using Trap x20 (GETC). Echo the characters to the console as they are received (OUT), and store them as **character** data in separate registers.

2. Output to the console the operation being performed e.g.
   5 - 7 =
   (how will you print the " - "? How will you print the " = "? Note the double quotes!!)

3. Once the setup is printed, convert the numeric characters into the actual numbers they represent (e.g. convert the *ASCII code for '7'* into the *binary representation of the number 7*).

4. Perform the subtraction operation *(by taking the two's complement of the second operand and adding)*, and determine the sign (+/-) of the result;
   if it is negative, determine the *magnitude* of the result *(i.e. take the 2's complement to turn it back into a positive number)*.

5. Convert the resulting *number* back to a printable *character* and output it, together with a minus sign if necessary. Remember, the number -4 when converted to text is actually two separate ascii characters, '-' and '4'.

Example, with a detailed algorithm *(we won't always give you this!)*

- Program prompts for user input (two characters):
- user enters ‘5’, which is echoed to console (followed by a newline) and copied to a register.
- user enters ‘7’, which is echoed to console (followed by a newline) and copied to a different register.
- Program outputs the text

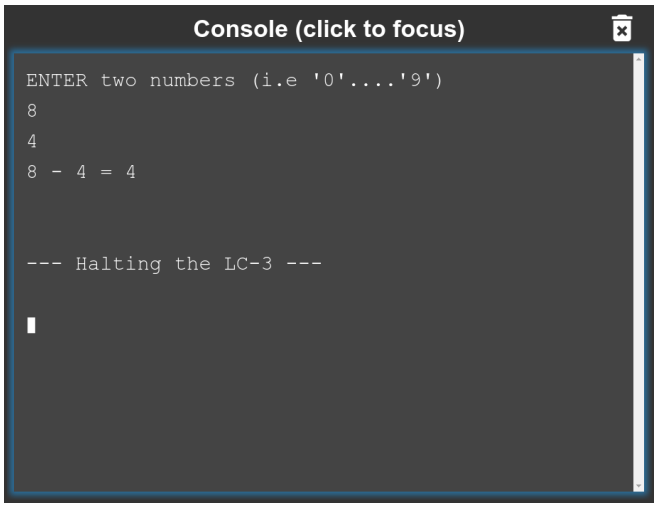
  ```
  5 - 7 =
  ```

  (this will actually require at least 4 distinct output steps using OUT and PUTS)
- Program converts ‘5’ *(ascii code)* into 5 *(number)* and stores it back in the same register.
- Program converts ‘7’ into 7 and stores it back in the same register.
- Program takes 2's complement of 7, and stores the result back into the same register.
- Program adds the contents of the two registers - i.e. it performs the operation (`5-7`) and stores the result (-2) in a third register.
- Program recognizes that the result is negative, obtains the magnitude of -2 (= 2), and outputs **'-'** (minus sign).
- Program converts 2 *(number)* into '2' *(ascii code)*, and stores it back in the same register.
- Program outputs '2' *followed by a newline*.

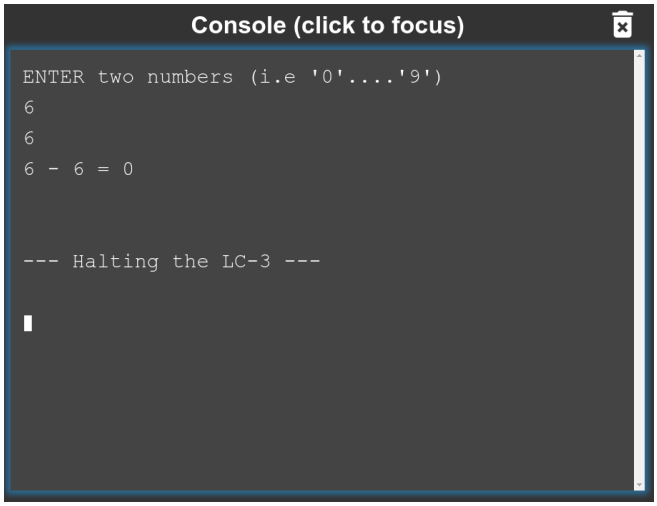
Expected/ Sample output

In this assignment, your output must ***exactly*** match the following, including:

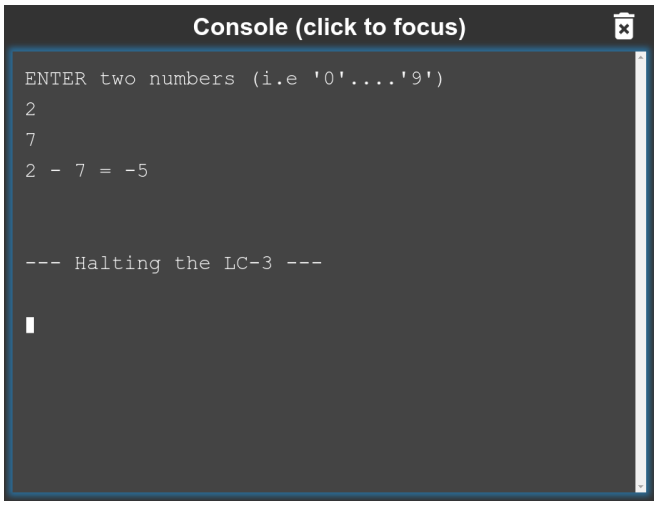
- the prompt, *followed by newline (provided in the starter code)*
- Each digit input "echoed" and *followed by a newline*
- the subtraction operation, including spaces as shown, also *followed by a newline*:



(Difference is Positive)



(Difference is Zero)



(Difference is Negative)

Your code will obviously be tested with a range of different operands giving all possible results. Make sure you test your code likewise!

NOTES:

- All console output must be **NEWLINE terminated.**
- We will test only with **positive single digit numeric inputs**
- **NO** error message is needed for invalid input (i.e. we will not test with non-numeric inputs)

Uh…help?

- Trap x20 (GETC) will *always* store the ascii code of the input character into ***R0***.
  You cannot specify any other register to receive the keyboard input.
- Trap x21 (OUT) will *always* print whatever ASCII code is currently stored in ***R0***. You cannot specify any other register to output to screen.
- If the user enters '7', the value stored into R0 is the ASCII code b0000 0000 0011 0111 (= x0037 = '7'), ***not*** the number 7 = b0000 0000 0000 0111 (= x0007 or #7).
  Likewise, to output the number 7 as a character, it has to be adjusted from x0007 to x0037 first.
  Go to www.asciitable.com or here and see why.

  Note: *you should ALWAYS refer to the ascii numeric offset as* ***x30****, NEVER as #48.*
  The ascii table only makes sense when understood as a ***binary*** *code (readable as hex).*

  Conversion between a character and the number it represents and vice versa will be used repeatedly in this course, so make sure you understand how to do it now!!
- To take the two's complement of a number *(i.e. make a positive number negative or vice versa)*:
  - Invert the bits *(what assembly instruction does this?)*
  - Add one
- A neat trick in LC3 to copy the value of one register directly to another:

```
ADD R5, R6, #0   ; R5 ← (R6) + 0,  i.e.  R5 ← (R6)
```

- If the result is negative, remember that you will have to print *two* characters, not one *(there is no ASCII code for '-1', right?)*
- If you are struggling with writing LC-3 code from scratch, try writing the program out in pseudo-code or even C++ first. Then, your only task is to convert the logic/code into LC-3.

Submission Instructions

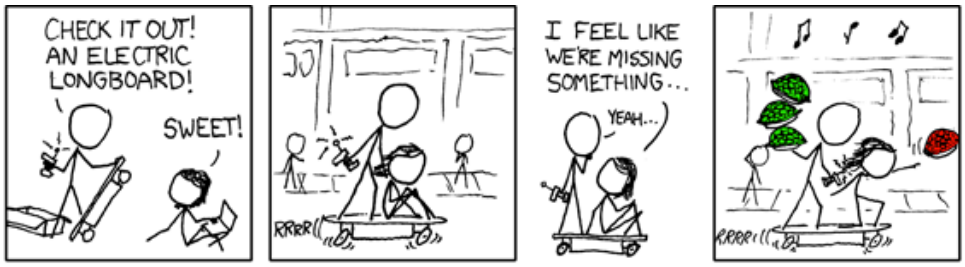
Submit ("Upload") your **assignment2.asm** file *(and ONLY that file!)* to the Programming Assignment 2 folder in Gradescope: the Autograder will run & report your grade within a minute or so.
You may submit as many times as you like - your grade will be that of your last submission.
If you wish to set 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.

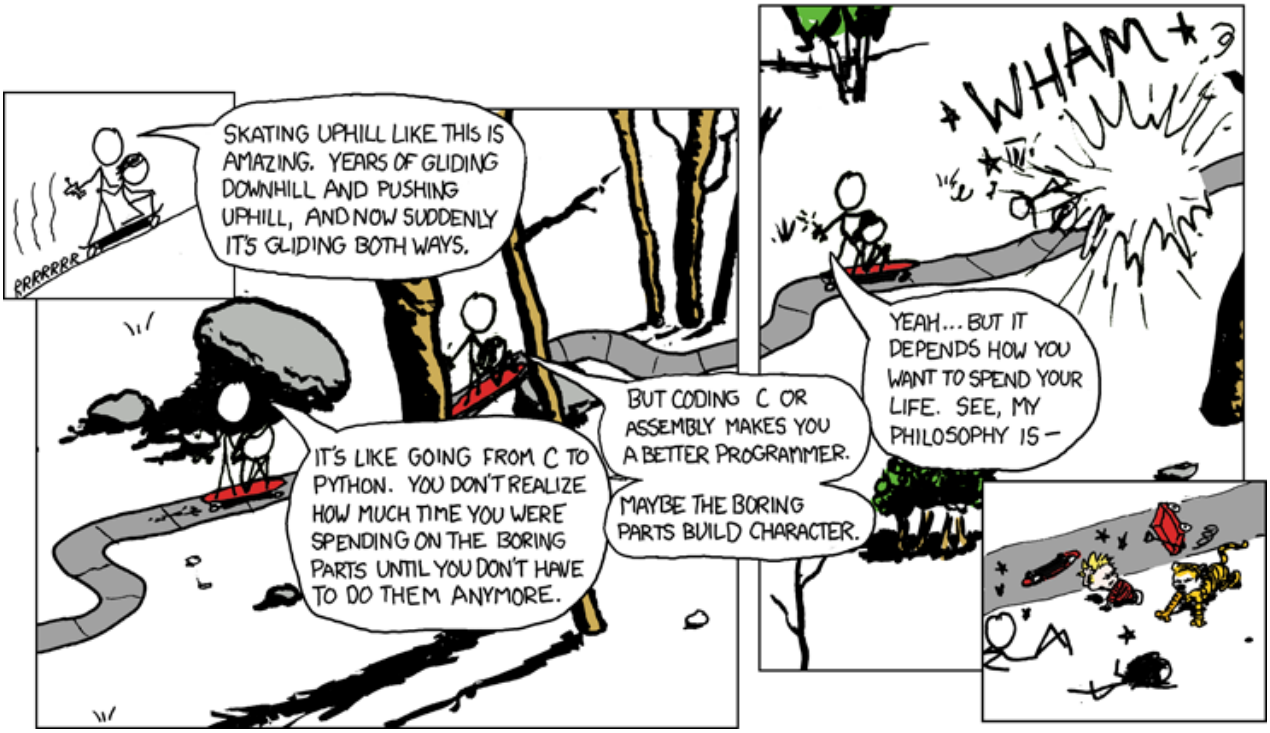
Rubric

- **To pass the assignment, you need a score of >= 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 => 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/409/