ELE209 Lab 6a
LC-3 Assembly Language Programming and Debugging

Due: You must email your code to the TA and demonstrate it by the **start** of lab the next week.

This lab is a two week lab. It consists of one large assembly language program which is broken into various subtasks. The subtasks must be completed and demonstrated to the TA in the order they are given.

Style requirements: You must follow the style guidelines given at the end of this assignment.

Assignment: You will write an assembly language program that will convert a number entered from the keyboard into a 16 bit two's compliment value, then display it to the console as a binary number.

The numbers have a form similar to how you enter values into registers in the LC-3 simulator. Specifically, they have the form

```
[SIGN][TYPE]number
```

where `SIGN` is an optional minus sign (ie. "-") and `TYPE` is an optional number type, where "x" means hexadecimal and "#" means decimal. If `TYPE` is not present, the type is assumed to be hexadecimal. Some examples are 123, -#456, xAB4, and -xFFFF.

A summary of the subtasks are as follows:

1. Read a null terminated string from keyboard into a buffer. (week 1)
2. Parse the non numeric part of the string to determine its sign and radix. (week 1)
3. Convert the numeric part of the string to a 16 bit twos compliment binary number. (week 2)
4. Output the 16 bit number to the console in binary. (week 2)

Task 1: Download `lab6a_1.asm` from the course website and complete it as follows.

Add code that reads a sequence of characters from the keyboard, converts any lowercase letters to uppercase, and stores them as a null terminated string in memory. Stop reading when either the enter key is pressed or the buffer is full. Do not include the enter key in your null terminated string.

Your code should be placed in the file where it says "`Insert your code here!`"

- Your code must start with a label named "ReadCore".
- On entry to your code,
  R1 will contain the length of the input buffer where you will store the null terminated string
  R2 will contain the address of this input buffer in memory
- On exit from your code,
  The input buffer given in R2 must contain a null terminated string of at most R1-1 characters.
- At the end of your code, it must unconditionally branch to a label called "ProcessInput".

Hint: You will probably want to use the `GETC` trap to read keys, and you will have to echo them yourself using the `OUT` trap. Also, it might be easier to implement it without case conversion first, then add it after the basic routine is working.

Demonstrate your working code to the TA before going on to part 2.

Task 2: Download `lab6a_2.asm` from the course website and complete it as follows.

Add code that parses the non-numeric part of the input string, and calls the appropriate parser for decimal or hexadecimal. You do not need to parse the number part yet.

Your code should be placed in the file where it says "`Insert your code for part 2 here!`". Also, you need to copy the code you wrote for part 1 to where it says `Insert your code from part 1 here!`". Copy only the code you wrote in part 1, do not copy the code that was already there.

- Your code must start with a label named "ProcessInputCore".
- On entry to your code,
  R2 will contain the address of a null terminated string containing the number to parse
- On exit from your code,
  R1 must contain zero if the number is positive and -1 if it is negative
  R2 must contain the memory location of the first digit of the number (ie. after the minus sign and "x" or "#").
- The code should branch to "ParseHex" if it determines that the number is hexadecimal, and "ParseDecimal" if it determines that it is decimal.

Style requirements:

Beyond making sure that your programs function correctly, it is important that they are documented thoroughly and correctly. Good programming "style" makes your programs easy to read and to understand, both by you and by anyone else who needs to understand or to change your code in some way. Furthermore, programs written following appropriate style guidelines are easier to write and debug than programs written in a haphazard, disorganized style. While the assembler does not care about the programming style you use, the goal of good style is to make the program easy for people to read.

1. Your assembly language programs should have the following comment block at the top of the file.

```
;
; Name(s):
; Course number:
; Lab number/problem number:
; Due date:
;
```

2. All symbolic labels must be meaningful. It is helpful to use both lower and upper case to help make the names easy to read. For example, use SaveR1 as a label instead of something more cryptic, such as SR1. Other good examples include names such as, EndProgram, ReadKeyboard, PrintResult, ShiftLeft, and so on.

3. Add a comment after each instruction to describe what it is doing with regard to the overall program.

4. Use white space (i.e., spaces, tabs, carriage returns) generously to emphasize the structure of your program and to make it easy to read. Make sure you indent and align statements and comments appropriately.

5. No lines should be more than 80 characters long.