CSE 6010
Final Assignment – Option 1
Carrying Out LC-3 Instructions
Choose Option 1 or Option 2 for your final project
If you submit both Option 1 and Option 2, the higher grade will be counted toward as your final project and the lower grade will count as the "Substitution Assignment" explained previously

NO Initial Submission
NO 48-hour grace period
Final Submission Due Date: 11:59pm on Monday, December 11
(Serves as a final exam; Monday, December 11 is our final exam day)
Submit codes as described herein to Gradescope through Canvas

In this assignment, you will write a program in C that will carry out a series of LC-3-like instructions. To do so, you will read in a file that contains memory addresses as 16-bit strings along with the 16-bit contents for each location. Some locations will contain data (for the LC-3, two's complement integers) and others instructions. The instructions will perform some operation; you will output the result that would be stored in memory.

Your code will need to be able to recognize and implement the following LC-3-like instructions; one will have a small variation, and one will be new as described. Opcodes, instruction formats, etc. will be as for the LC-3 unless specified otherwise.

- LD: load to a register the value at an address calculated a specified offset from the PC.
- LDR: load to a register the value at an address calculated a specified offset from the value of a base register.
- ADD: store to a register the sum of the values of two source registers.
- ADD: store to a register the sum of the value of one source register and an immediate operand.
- BRp: branch that works the same as BRz that we discussed extensively but testing the p bit, the third of the condition codes NZP.
- STR: normally store to a register the value at an address calculated a specified offset from the value of a base register; in this case, as there will be no further use of an item that would be stored back to memory, you will simply output to the screen the address and the value that would be stored. No more than one value will be saved using STR.
- FIN: a new instruction with opcode 1101 (which was unassigned in the LC-3 instruction set), indicating that that instruction is the last of the program encoded in the instructions stored in memory.

Note that all addition (whether in the ADD instruction or in calculating addresses for other instructions) involves two's complement numbers of signed integers; operands may be negative.

Because you will need to work with strings, you may need to research and learn new string functions. Recall that a string is a character array. In addition, recall that a string should include one extra character known as the ***"end of string" character*** that can be specified as `\0`. Here are a few tips that should help you get started (there are plenty of sources online where you can learn the syntax/usage!).

- To use many string functions, you will need to `#include <string.h>`.
- The function `strncpy()` can be used to copy up to n characters from a source string to a specified destination.
- The function `strtol()` can be used to convert a string to a long int in a specified base.

You may wish to convert bit strings to integers to perform various computations. You are not expected to write code that directly adds the 16-bit representations of two integers.

You may test your code against the **autograder** for the final submission; you may submit as many times as you like.

Specifications:

- Input and command-line argument: Your program should take 1 required command-line argument, which is the filename that specifies the memory addresses for the relevant instructions and data as well as the values stored at those locations. The first line of the file will include one integer that specifies the number of memory locations occupied. Each remaining line corresponds to a memory location and its contents. These items consist of two 16-bit strings (0's and 1's) separated by a space, with the first representing the memory address and the second the contents of memory at that address. You may assume that a valid filename is specified and are not required to perform any validation. Two sample input files are provided.
- Setup:
    - You will need to represent main memory. Because there will not be much in memory (only what is specified in the input file), you may choose to use a linked list or a small array that stores memory addresses as well as data. Of course, you are also welcome to use a large array indexed according to memory address. Keep in mind that because the LC-3 uses 16-bit addresses, such an array would need 2^{16} entries.
    - For each item in memory, you will need to store the 16-bit string. You may also find it useful to store base-10 representations of the data and/or address. Keep in mind that the base-10 representation of a 16-bit instruction does not have any particular meaning, but it can be calculated.
    - You will need to have representations of the general-purpose registers so that instructions may utilize these values. Assume that all register values are automatically initialized to 0 when the program you are simulating begins. However, you will need to perform this initialization in your code.
    - You will need to have a representation of condition codes, as they will be checked during BR instructions. Remember that condition codes should be updated any time a value is stored in a general-purpose register, according to the sign of that value.
    - You will need to have a representation of the PC to advance to the next instruction.
    - You may wish to write a function to print the register values to aid in your development/testing.
- Output and cleanup:
    - Your code should output a single line that looks something like the following:
      `A03 22`

That is, you should output the memory location in hexadecimal, followed by the value that would be written as a signed int. You can output in hexadecimal using the format specifier %X (you do not need to perform any conversions, nor do you need to fill in any leading 0's).

- Be sure to free any dynamically allocated memory.

To receive full credit, your code must be well structured and documented so that it is easy to understand. Be sure to include comments that explain your code statements and structure.

A test input file is provided for you; we will check results for other files.

Submission: You should submit to the "Instructions" Final Project option in Gradescope through Canvas the following files:

(1) your code (all .c and .h files); no specific filenames are required for your code but ***note the specification for your executable file below***.

(2) the **Makefile** you use to compile your program such that your executable file that will run your program is named '**instructions**'. You may use the Makefile provided for previous assignments as a starting place. **Don't forget to include any needed compiler flags!**

(3) a series of 2-3 slides **saved as a PDF** and named **slides.pdf**, structured as follows:

- Slide 1: your name and a brief explanation of how you developed/structured your program. This should not be a recitation of material included in this assignment document but should focus on the main structural and functional elements of your program (e.g., the purpose of any loops you used, the purpose of any if statements you used to change the flow of the program, the purpose of any functions you created, etc.). The goal is to help us understand how you developed your code and your intentions. You are limited to one slide.
- Slides 2-3: a discussion of why you believe your program functions properly.

Some hints:

- Start early!
- Identify a logical sequence for implementing the desired functionality. Then implement one piece at a time and verify each piece works properly before proceeding to implement the next.