CS 1428 Honors
Lab 8

Jared Wallace

Overview

Today we will be extending our assembler to make it Turing Complete. We are going to modify it so that the assembler now incorporates a *jump* instruction and a *conditional jump* instruction. Additionally, we will be *refactoring* the assembler to utilize functions. Refactoring is the process of modifying code so that the internal structure changes, but the external behavior remains consistant. For our assembler, this will largely consist of simple cut and paste actions, but in the real world it usually involves changing algorithms to be more efficient and other, more substantive changes.

Questions

1. (15 pts) Last week, we had modified our assembler to read in the instructions for each program into "program" memory. When we did that, each instruction line gained an additional piece of associated data – its *index* in the program array. This index number functions as a line number does in a traditional program. Accordingly, just as you can specify a line number in C++ to "goto", we wish to implement jumping to a certain index number for our assembler.

   If you recall, our assembler reads through the instruction lines by using a loop. We can therefore implement our "jump" instruction by manipulating the loops index variable. *(Usually the index variable is 'i').* Be careful! In a for loop, the index variable is updated, or incremented, when the for loop reaches the closing bracket. You must account for this in your jump function, lest you incur an "off by one" error. For now, write a simple case statement to correctly handle the jump command. Use the following format:

   ```
   8 [offset] 0 0
   ```

   Hint: offset values can be negative to effect a jump "backwards"

2. (15 pts) Our conditional jump instruction will take the form of a "Branch if Equal" statement. In other words, the instruction will have three values: the offset and two index values of data to compare. If the two data values are equal, the jump occurs to the specified offset. Otherwise, nothing changes and execution continues at the next instruction. The instruction will take the following form:

```
9 [offset] [index 1] [index 2]
```

   Write the case statement implementing this feature.

3. (70 pts) Using the answers from questions one through two, modify your assembler so that it implements both *jump* and the *conditional jump*. Your program should comply with the following requirements:

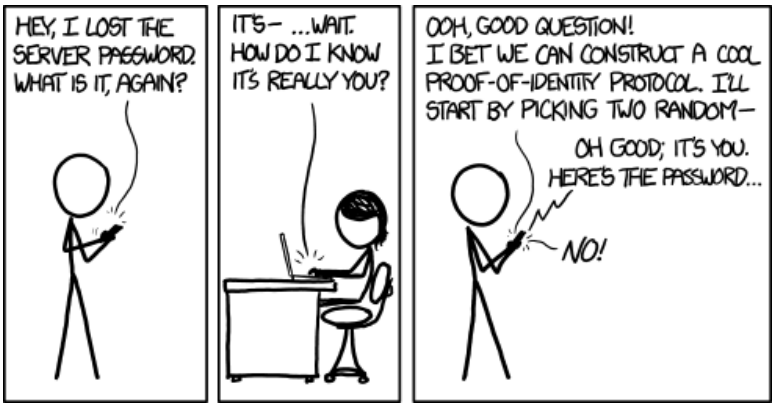
   - Reads all instructions into memory at one time.
   - Processes the instructions in order, one line at a time.
   - Properly handles the same sample programs from last time (supplied)
   - Properly handles one new program that includes jumps and conditional jumps
   - Implements each type of instruction as its own function, as well as creating a function to handle the initial reading and storage of instructions into program memory. Use the following prototype for this:

```
int loadProgram(string filename, int program_memory[512][4]);
```

     The value of the integer returned should be the number of lines of instructions read into memory. If the operation fails, it should return 0.

Deliverables

Hard copy of the source code you wrote (assembler.cpp). Soft copy (upload to homework upload) of your source code. As always, make sure you make a commit and push your work.



Not sure why I just taught everyone to flawlessly impersonate me to pretty much anyone I know. Just remember to constantly bring up how cool it is that birds are dinosaurs and you'll be set.