CS2110 Fall 2017

Homework 6

Author: Patrick Tam, Mert Dumenci, and Madeleine Brickell

Objectives

The goal of this assignment is to help you become comfortable coding in the LC-3 assembly language. This will involve creating small programs, printing to the console, and converting simple algorithms into assembly code.

Overview

A Few Requirements

1. Your code must assemble with **NO WARNINGS OR ERRORS**. To assemble your program, open the file with Complx. It will complain if there are any issues. **If your code does not assemble you WILL get a zero for that file.**

2. Comment your code! This is especially important in assembly, because it's much harder to interpret what is happening later, and you'll be glad you left yourself notes on what certain instructions are contributing to the code. Comment things like what registers are being used for and what not so intuitive lines of code are actually doing. To comment code in LC-3 assembly just type a semicolon (;), and the rest of that line will be a comment.

   Avoid stating the obvious in your comments, it doesn't help in understanding what the code is doing.

   Good Comment
   ```
   ADD R3, R3, -1                ; counter--
   BRp LOOP                      ; if counter == 0 don't loop again
   ```

   Bad Comment
   ```
   ADD R3, R3, -1                ; Decrement
   R3 BRp LOOP                   ; Branch to LOOP if positive
   ```

3. **DO NOT assume that ANYTHING in the LC-3 is already zero.** Treat the machine as if your program was loaded into a machine with random values stored in the memory and register file.

4. Following from 3. You can randomize memory by using the menu option State → Randomize. And then load your program by saying File → Load Over.

5. You may NOT use the instructions JMP, JSR, or JSRR. Do not write any TRAPs.

6. Do NOT execute any data as if it were an instruction (meaning you should put .fills after halt).

7. Do not add any comments beginning with @plugin or change any comments of this kind.

8. Test your assembly. Don't just assume it works and turn it in.

Assembly Overview

For this assignment, you will be writing 3 assembly programs. The purpose of this assignment is to get you familiar and more comfortable with writing low-level assembly code. For each part of this homework, we will supply you with an empty framework file containing the description of a single algorithm or program. You must implement these programs in assembly.

```
.orig x3000
      LEA R0, HW ;Load the address of the string
      PUTS       ;Output the string
      HALT       ;Stop Running
      HW .stringz "Hello World.\n"
.end
```

This is a simple program that prints "Hello World." and a new line to the console.

> **;** denotes a comment. You don't need a semicolon after every line, only before comments.
>
> .orig is a pseudo-op. Pseudo-ops are special instructions for the assembler that are not actually assembly instructions. ".orig x3000" tells the assembler to place this block of code at x3000, which is where the LC-3 starts code execution. Therefore "LEA R0, HW" is at address x3000, "PUTS" is at address x3001, etc.

Next is your assembly program. You've seen this before. PUTS is just a pseudonym for a TRAP that prints a string whose address is stored in R0, and HALT is a TRAP that stops the LC-3.

> .stringz is another pseudo-op the stores the following string at that set of memory locations, followed by a zero (That's what the 'z' is for). For example, 'H' is stored at x3003, 'e' is stored at x3004, etc.
>
> .end tells the assembler where to stop reading code for the current code block. Every .orig statement must have a corresponding .end statement.

Other pseudo-ops you should know are:

> .fill [value] - put the given value at that memory location (.fill x6000, .fill 7, etc)
> .blkw [n] - reserve the next n memory locations, filling them with 0.

After writing your assembly code, you can test it using Complx. Complx is a full featured LC-3 simulator, and we recommend running your code in the following fashion:

1. Go to the "State" menu and click on "Randomize". This randomly assigns values to every memory location and register, and prevents you from assuming values will be initialized to 0.

2. Go to the "File" menu, select "Load Over" and browse for your assembly file. This will load your code over the randomized memory. You will see your code load in the main window.

3. From here, you can run your code. Click the "Run" button to run your code automatically until a HALT instruction or breakpoint is hit. Click "Step" to execute one instruction at a time. Clickm "Next Line" to fast-forward through subroutines. Notice you can also step back.

One more thing, it is very important that you use Appendix A in the book to your advantage. Make sure you know what the instruction does before you use it. Think about how you can set a register with a specific number, how can you load a value from a label, how can you compare if two numbers are equal, and so on.

Take a look at this instruction: **LD R2, 5**

What exactly is this instruction doing? Is it loading the number 5 into R2? Remember that the bits preceding the destination register are based on the PCOFFSET9, this is not an immediate value, you are merely loading data from an address location.

Summary of the rules

Failure to follow these will result in a heavy point deduction, if not a zero, for this part.

1. Your program must assembly without errors.
2. Comment your code.
3. Do not assume any registers or memory are initialized to zero. Use the "Randomize" menu to option to test this.
4. Do not use the JMP, JSR, or JSRR instructions. Do not use TRAPs.
5. Test your code with the provide tests before submitting.

Deliverables

bizzfuzz.asm
insert.asm
sum.asm

After you have submitted your assignment make sure to redownload it and make sure that it passes all the tests. You alone are responsible for what you submit, and you risk getting a zero if you submit the wrong files, or forget some. **You have been warned!**

The Assignment

For each of the .asm files given, complete the program described in the comments at the top of each file. The program descriptions are copied here for your convenience.

Part 0 - Installing Complx

Before starting this homework, you must install Complx either natively on Windows/Mac/Linux, or in a Linux VM. After installing Complx, you must show your recitation TA that you have successfully done so during the lab on Wednesday, October 4th 2017. If you do not have Complx installed by the lab, you can come into office hours before end of day Friday and get checked off.

Part 1 - (sum.asm)

```
Write a program sums the elements in an array
```

1. Store the sum in the label SUM once the program has terminated.
2. If SUM is not initially zero, increment SUM by the sum of the array.
3. The array's length is stored in the label LENGTH
4. The address of the array's first element is stored in the label ARRAY

Part 2 - (insert.asm)

```
Write a program to add an element at a certain index in an array. For the
tester, it checks (element, index, [array])
```

1. Do not delete any elements from the array, just shift elements when adding
2. Elements can be inserted at the front, middle, or back of the array
3. The array begins at index 0

Part 3 - (bizzfuzz.asm)

```
Write a program to do the following.

Given an array of positive integers (`ARRAY`) of length `LENGTH`, for each
integer `n` inside the array:
```

1. Print 'Bizz' and a newline if `n` is divisible by 3 and not by 5 (`n mod 3 == 0` and `n mod 5 != 0`)
2. Print 'Fuzz' and a newline if `n` is divisible by 5 and not by 3 (`n mod 3 != 0` and `n mod 5 == 0`)
3. Print 'BizzFuzz' and a newline if `n` is divisible by 3 and 5. (`n mod 3 == 0` and `n mod 5 == 0`)

Rules and Regulations

General Rules

1. Starting with the assembly homeworks, Any code you write (if any) must be clearly commented and the comments must be meaningful. You should comment your code in terms of the algorithm you are implementing we all know what the line of code does.

2. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.

3. Please read the assignment in its entirety before asking questions.

4. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.

5. If you find any problems with the assignment it would be greatly appreciated if you reported them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere in the file, near the top, for other files unless otherwise noted.

2. When preparing your submission you may either submit the files individually to T-Square or you may submit an archive (zip or tar.gz only please) of the files (preferred). You can create an archive by right clicking on files and selecting the appropriate compress option on your system.

3. If you choose to submit an archive please don't zip up a folder with the files, only submit an archive of the files we want. (See **Deliverables**).

4. Do not submit compiled files that is .class files for Java code and .o files for C code. Only submit the files we ask for in the assignment.

5. Do not submit links to files. We will not grade assignments submitted this way as it is easy to change the files after the submission period ends.

Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know ***IN ADVANCE*** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. After submitting you should be sure to download your submission into a brand new folder and test if it works. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via Canvas. When you submit the assignment you should get an email from Canvas telling you that you submitted the assignment. If you do not get this email that means that you did not complete the submission process correctly. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over Canvas.

3. There is a 6-hour grace period added to all assignments. You may submit your assignment without penalty up until 11:55PM, or with 25% penalty up until 5:55AM. *So what you should take from this is not to start assignments on the last day and plan to submit right at 11:54AM.* You alone are responsible for submitting your homework before the grace period begins or ends; neither Canvas, nor your flaky internet are to blame if you are unable to submit because you banked on your computer working up until 11:54PM. The penalty for submitting during the grace period (25%) or after (no credit) is non-negotiable.

Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class.
Quizzes, timed labs and the final examination are individual work.

Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using electronic computer programs to find evidence of unauthorized collaboration.

What is unauthorized collaboration? Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment. Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student via electronic means. This includes simply e-mailing it to them so they can look at it. If you supply an electronic copy of your homework to another student and they are charged with copying, you will also be charged. This includes storing your code on any site which would allow other parties to obtain your code such as but not limited to public repositories (Github), pastebin, etc. If you would like to use version control, use github.gatech.edu

Is collaboration allowed?

Collaboration is allowed on a high level, meaning that you may discuss design points and concepts relevant to the homework with your peers, as well as help each other debug code. What you shouldn't be doing, however, is paired programming where you collaborate with each other on a low level. Furthermore, sending an electronic copy of your homework to another student for them to look at and figure out what is wrong with their code is not an acceptable way to help them, and it is often the case that the recipient will simply modify the code and submit it as their own.