CS2110 Summer 2016
Homework 05

Author: Austin Herring

This assignment is due by:

Day: June 21, 2015
Time: 11:55:00pm

Rules and Regulations

Academic Misconduct

Academic misconduct is taken very seriously in this class. Homework assignments are collaborative. However, each of these assignments should be coded by you and only you. This means you may not copy code from your peers, someone who has already taken this course, or from the Internet. You may work with others **who are enrolled in the course,** but each student should be turning in their own version of the assignment. Be very careful when supplying your work to a classmate that promises just to look at it. If he/she turns it in as his own you will both be charged.

We will be using automated code analysis and comparison tools to enforce these rules. **If you are caught you will receive a zero and will be reported to Dean of Students.**

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 T-Square. When you submit the assignment you should get an email from T-Square 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 T-Square.

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 T-Square, 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.

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.

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. 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, etc.

Overview

The goal of this assignment is to help you become comfortable with the different components of the LC-3 and capable of coding in the LC-3 assembly language. This will involve the creating small programs, reading input from the keyboard, printing to the console, and converting from high-level code to assembly. The entire assignment is composed of two main parts.

1. A multiple choice quiz.
2. Writing your own assembly programs that mimic a segment of C code provided to you.

Part 1:
Introduction to LC-3 and Assembly

For the first part of the assignment, you will need to read chapters 4, 5, and 7 of the text book. In addition to reading these chapters, you will be required to install a distribution of Linux and an LC-3 Simulator (Complx). If you attended lab Tuesday, then you should be good to go! If not, the last section of this document explains the steps required to set up. Once you have completed these two things you may precede.

The quiz that you will take is located on T-square under “Tests & Quizzes.” Even though this is in the “quizzes” section, it will still be graded as a homework. You will have unlimited attempts to complete the assessment, but no feedback on how you did. This means you must be confident in your answers! Obviously, you can also come to office hours for help too. It will cover questions on Tracing LC-3 Instructions / Converting Assembly to Hex / The LC-3 Datapath.

Once you have completed the quiz, you may move on to part 2.

Part 2:
Writing Assembly Programs

For this part of the assignment, you will be writing 2 assembly programs. The purpose of this is to get your familiar and more comfortable with writing low-level assembly code. For each program of this homework, we will supply you with pseudo-code, and you must write the equivalent assembly code. Please make your code as similar to the pseudo-code as you can!

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 left notes to let you know sections of code or 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 semi-colon (;), 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.
Test your assembly. Don't just assume it works and turn it in.

An example

You've been given framework files for this assignment, but I'll tell you a little bit about how the assembly files in this class look.

```
.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 assembly program that prints “Hello World.” and a new line to the console.

; denotes a comment. You don't need a semi-colon 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.

Testing and Additional Notes

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. Click "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 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.

Program 1 – deMorgan's Law (demorgans.asm)

In LC-3 assembly you will notice that there is no OR operation. But that can easily be fixed by using deMorgan's Law. Your task is to store the value A | B in your assembly program. deMorgan's law: ~(~A & ~B) == (A | B). **Make sure you test you solution with different values.**

Here's the pseudo-code:

```
/* A and B are parameters, answer is the return
A = !A;
B = !B;
answer = A & B;
answer = !answer;
return answer;
```

A and B are labels for a .fill in the template file. You will need to load the values A and B into a register.
ANSWER is a label for another .fill in the template file. You will need to store the value you get for answer at the address labeled ANSWER.

A and B will be any integer in the range -32767 to 32767, inclusive.

Program 2 – History Timeline (timeline.asm)

Its time for a history review! It has probably been a little while since you took a history course, so we decided to throw in a nice refresher alongside with your assembly practice. Here is what we want you to do. Given a year, tell us the important dates preceding it (**moving further back in history**) and tell us the important dates succeeding it (**moving forward in history**). You will tell us these dates by printing out lines to the Complx. The function will take in 'N', a number between 1 and 10 inclusively. N will then be used to index `YEAR_ARR` and `EVENT_ARR`. This table visualizes the arrays and how to index them.

N	YEAR_ARR	EVENT_ARR
1	1607	John Smith founded Jamestown in 1607
2	1776	The Declaration of Independence was signed in 1776
3	1788	The Constitution was ratified in 1788
4	1861	The Civil War Broke out in 1861
5	1879	Thomas Edison invented the light bulb in 1879
6	1917	The US entered WWI in 1917
7	1941	Japan attacked Pearl Harbor in 1941
8	1961	The Vietnam War began in 1961
9	1969	Apollo 11 landed on the moon in 1969
10	1985	Super Mario Bros. debuted in 1985

Here is the pseudo-code:

```
void timeline(int n){
     n--;      //Zero index N
     if (n==0) {
          printf(“Nothing happened before “);
          printf(YEAR_ARR[n] + “\n”);
     } else {
          printf(“In the years before “);
          printf(YEAR_ARR[n] + “\n”);
          int i = n;
          while( i > 0) {
               i--;
               printf(EVENT_ARR[i]);
          }
     }

     if (n==9) {
          printf(“And nothing happened after “);
          printf(YEAR_ARR[n] + “\n”);
     } else {
          printf(“And in the years after “);
          printf(YEAR_ARR[n] + “\n”);
          int i = n;
          while( i < 9) {
               i++;
               printf(EVENT_ARR[i]);
          }
     }

     return;
}
```

Here, `YEAR_ARR` and `EVENT_ARR` are arrays with pointers to different .stringz values. `YEAR_ARR` contains the pointers to the years and `EVENT_ARR` contains pointers to the corresponding events. These values are all supplied with the template file.

For example, if the value of N is 4 then the output should be:

```
In the years before 1861
The Constitution was ratified in 1788
The Declaration of Independence was signed in 1776
John Smith founded Jamestown in 1607
And in the years after 1861
Thomas Edison invented the light bulb in 1879
The US entered WWI in 1917
...Continues...
Super Mario Bros. debuted in 1985
```

Don't forget the edges case either! For example, N=1. Make sure to look at the pseudo-code!

Hint: Because `YEAR_ARR` and `EVENT_ARR` are arrays containing pointers to strings, you cannot simply offset the array index by the day and print what is there (since what is there is a pointer). It’s a step in the right direction though.

Deliverables

- The Assessment, HW05, is completed on T-Square under “Tests & Quizzes”
- demorgans.asm
- timeline.asm

Remember to put your name at the top of EACH .asm file you submit. The files should be named exactly as given.

Linux / Complex Install Instructions

The Linux distribution we will be using is Ubuntu 15.04 64-bit. We highly recommend you use this version of Ubuntu because it is the most up to date and includes many libraries we will use in the default repositories. You are allowed to use older versions of Ubuntu or other distros, but installing the tools and programs for this course will be more complicated or may not work at all. The TAs may not be able to help you install distributions other than Ubuntu 15.04 64-bit. The only Linux distro we guarantee support for is Ubuntu 15.04 64-bit. If you are installing a Debian based distro, the majority of the TAs can provide some level of support. If you are installing a Non-Debian based distro, Collin, Nathan and Andrew are the people to talk to.

Linux is an operating system like Windows or OSX. It needs to believe that it has its own hardware that it is running on. There are a couple different ways of achieving this. We recommend virtualization. Virtualization emulates the hardware of a computer in the software of another operating system. This means that you can run a secondary operating system inside of your primary operating system. Since virtualization is running a program inside of an operating system inside of an operating system, it can be slow. The main advantage of virtualization is it is safe. We highly recommend virtualizing Linux because you do not run the risk of losing your data. The software we recommend for virtualization is VMware Player/Fusion or Virtualbox.

Another way to install Linux is to install it on your computer in parallel to your main operating system. When you boot up your computer, you can choose which operating system you will use. This is called dual booting. You are allowed to dual boot Linux if you want, but DO SO AT YOUR OWN RISK. Additionally, we the TA's will not be available to assist you. Installing Linux along side Windows/OSX runs the risk of destroying your Windows/OSX partition and losing your data if you don't know what you're doing.

Lab on Monday will be a help session for installing Linux and Complx. In order to complete Homework 6, you will need to have Complx installed.

TL;DR Install VMware Player/Fusion or Virtualbox, then install Ubuntu 15.04 64-bit, then install Complx.

Installing Ubuntu 15.04 64-bit with VMware Player/Fusion:

1) Download and install VMware Player (Windows), VMware Fusion (OSX), or Virtualbox (Windows/OSX). For Windows, we recommend using VMware Player. For OSX, we recommend Virtualbox. In past semesters, Georgia Tech used to provide VMware Fusion for free through dreamspark, though that may not be the case this semester.

2) Download Ubuntu 64-bit: http://www.ubuntu.com/download/desktop

3) Create your virtual machine using either VMware Player/Fusion or Virtualbox. This process is relatively simple with both programs. They will both create a virtual machine and ask for the .iso file to boot from. When creating the virtual harddrive, we recommend you create a fixed size harddrive of 15-20gigs. When first booting, choose the Ubuntu file you downloaded and it will boot into a linux live session (no changes will be saved until you install). On the desktop should be a program to install Ubuntu. When installing, choose the option to erase entire disk and

install ubuntu; what it will "erase" is the virtual harddrive that you made (which is empty). Run through it and you will have Ubuntu installed!

3b) You may need to change a few settings to get your virtual machine to run well. Regardless of the program you used, if things are running slow, you can increase the number of cpu cores and memory allocated. For instance, I'm running my virtual machine with 2 cpu cores (out of the 4 on my machine) and 1 gig of ram (out of 8). IF YOU ARE USING VIRTUALBOX, enable 3D Video acceleration.

4) Once you have installed Ubuntu, open the terminal by either opening the menu with the Windows Key/Command and typing "terminal" or pressing control+alt+t on your keyboard. Enter "sudo apt-get install nautilus-open-terminal". When that is done running, reboot your virtual machine. This will add an option "Open Terminal Here" when you right click inside of or on a folder. This will prove to be very useful for the rest of the class.

Installing Complx, the LC3 simulator we use for 2110:

1) Now that you have Linux up and running, we need to build and install the LC3 simulator we will be using for the class, Complx. Go to https://github.com/TricksterGuy/complx and download the source with the "Download zip" button on the right. Extract the contents of the zip file.

2) If you are not using Ubuntu 15.04, refer to the README-BUILD-INSTRUCTIONS.txt file for further instructions on how to install Complx. If you have Ubuntu 15.04, continue following the instructions

3) Right click on the complx-master folder and select "Open in terminal" to create a new terminal in the complx-master directory. Here we're going to be installing a few packages before we build and install Complx. Run the following commands (you can paste things to terminal windows using Shift+Control+V). If prompted, enter y to confirm that you want to install the packages:

```
sudo apt-get install build-essential
sudo apt-get install libwxgtk3.0-dev
```

4) Now that we have the dependencies and build tools installed, run the following commands to build and install complx:

```
sudo make install
sudo ldconfig
```

5) Test to make sure that Complx install correctly by running the command "complx" in terminal. If the program opens up, then everything is working! You can delete the complx-master folder and zip if you want; you don't need them any more.