CMPE 12L Lab 3 - Fall 2014
Writing Assembly and Generating Machine Code

Prof. Matthew Guthaus
Due: October 31, 2014 5pm
75 Points (20 Report, 55 Work)

Prerequisites

- Read through this **entire** lab assignment.
- Read the textbook chapter 5 and 6.
- Review the lecture notes on the LC3 architecture.

Tutor/TA Review

Your lab tutor/TA will cover the following items in the first portion of the first lab:

- How to run LC3Edit and Simulate
- The GUI of Simulate
- How to step through a program in Simulate
- What's required

Overview

You are asked to write a program in LC-3 assembly language that takes a 16-bit (2 byte) word stored at memory location x3100. It uses this as the number of words to search starting at address x3200 while summing the total number of 1's in all the words. In the end, you will store the number of 1's as a 16-bit word at address x3101. For example: if the value x0002 is stored at memory location x3100, your program will read 2 words in memory such as:

```
x3200 0000100000000011
x3201 0000000010000001
```

(addresses in hex and data in binary for convenience). It will then store the value x0005 into memory location x3101 since there are five 1's in those two data words.

How should you approach this problem? First, you need to write a loop that loads words that are in adjacent addresses from x3200 to x3200+n where n was the value stored at x3100. With what jump or branch instruction can you use create the loop? How do you decide when to stop the branch/jump? How do you decide where to jump/branch?

Once you load a word, you need to count the number of 1's. ADDing a bit pattern to itself effectively shifts the bit pattern to the left by one bit position. If I have a number X and I AND it with the bit pattern 0000000000000001, what do I get? What do I get when I AND the same number with 0000000000000010?

HINT: When you write your program, first write the assembly insructions and then convert it into machine language. It is difficult to read an entire program written machine language, but it is easier to "check" if an instruction is encoded right. Leave both the assembly and binary machine language in your file like this:

```
;; LEA R0, x002
1110 000 000000010
```

where the lines beginning with a semicolon are actually comments. You can also put spaces to divide up the opcode, registers, and immediate values for easy modification later.

In this lab, you will use the LC3 simulator to verify that your program works correctly. This is a processor simulator that implements a full instruction set architecture (ISA) and uses an ALU (with many more operations) like the one that you designed in lab 2. LC3 is a free simulation tool and is already installed on the lab machines, but if you would like to get the tool for home (you still have to come to lab), it is available at

`http://users.soe.ucsc.edu/~mrg/LC301.exe`

for Windows and at

`http://users.soe.ucsc.edu/~mrg/lc3tools_v12.zip`

as source code for a variety of Unix-based machines (including Mac OSX). This document assumes the Windows tools – you should be able to figure out the Unix-based tools on your own. You will use the LC-3 tools to simulate your programs to better understand how a processor works and to verify that your program is correct.

The LC-3 Editor

- Start the LC-3 editor, LC3Edit.exe. You can find it on the PCServer in $F:\backslash ClassFolders\backslash ComputerEngineering\backslash CE12L\backslash LC3$.
- Copy the hello.bin sample code in Figure 1 into the LC3Edit program.
- Convert the code from ASCII binary (1 and 0 characters) to true binary by selecting clicking on the "B" button (

). Save the file in your mounted home directory ($X:\backslash$).

```
;; CMPE12 - Fall 2014
;; hello.bin
;; This program echoes the text "Hello, world" to the output.

;; The code will begin in memory at address x3000
0011 0000 0000 0000
;;  load the greeting address
;; LEA R0, x002
1110 000 000000010
;;  print out the greeting
;; TRAP x22
1111 0000 00100010
;; stop the processor
;; TRAP x25
1111 0000 00100101
;; data declarations from here on
0000 0000 0100 1000 ; H
0000 0000 0110 0101 ; e
0000 0000 0110 1100 ; l
0000 0000 0110 1100 ; l
0000 0000 0110 1111 ; o
0000 0000 0010 1100 ; ,
0000 0000 0010 0000 ; <space>
0000 0000 0111 0111 ; w
0000 0000 0110 1111 ; o
0000 0000 0111 0010 ; r
0000 0000 0110 1100 ; l
0000 0000 0110 0100 ; d
0000 0000 0000 1010 ; newline
0000 0000 0000 0000 ; null
```

Figure 1: Machine code program to display "Hello, world": hello.bin.

- Check for errors. If there were problems converting, correct the errors now.
- Upon successful conversion, you should have generated the object file hello.obj. This can be run in the simulator.

The LC-3 Simulator

For the next two lab assignments, you will use Simulate to simulate an LC-3 processor.

- Start the LC-3 simulator, Simulate.exe. You can find it on the PCServer in $F:\backslash ClassFolders\backslash ComputerEngineering\backslash CE12L\backslash LC3$.
- In the simulator, open your object (.obj) file. You can open it with the File->Load Program dialog, or with the toolbar button as shown in Figure 2.
- Now, you are ready to run the program. You can use the Run Program button in the toolbar (see Figure 3) to run your program all the way through. The screenshot in Figure 4 shows another program having been run once through.

- You can rerun your program by selecting a new starting address. If you just hit Run without resetting the starting address, the simulator will yell at you! Reset the starting address by setting the "Jump to" field to x3000, and set the program counter (PC) to that address by pushing the button with the blue arrow. Recall that x3000 is the starting address of the program (the first two bytes in the object file are x3000). Also, you can use the step-into and step-over buttons (next to the Run button) your program to see each line of code being executed one at a time. See Figure 5.
- Every time you make changes to the machine code, you must re-convert it, re-load it in the simulator, and run it, verifying that it works as expected.

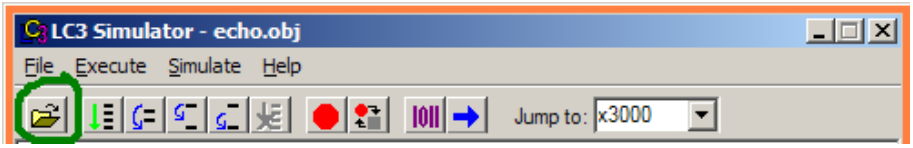

Figure 2: Use this toolbar button to open an object in the simulator

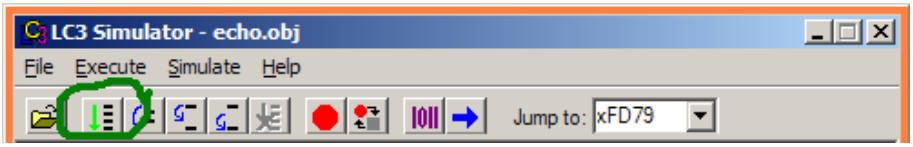

Figure 3: Use this toolbar button to run an object in the simulator.

Lab Submission

Your lab will be submitted via your eCommons account. Please log in to eCommons using your UCSC account and attach the following files to your "Lab3" assignment submission:

- lab3.bin
- lab3_report.pdf

Note that the final report must be submitted in PDF format. Make sure to confirm that your assignment is SAVED and SUBMITTED before the deadline. You may resubmit your assignment an unlimited number of times up until the due date.

Check-off

For this lab, as with most labs, you will need to demonstrate your lab when it is finished to a TA/tutor and get it signed off. This must be done in any of your lab sections BEFORE the final

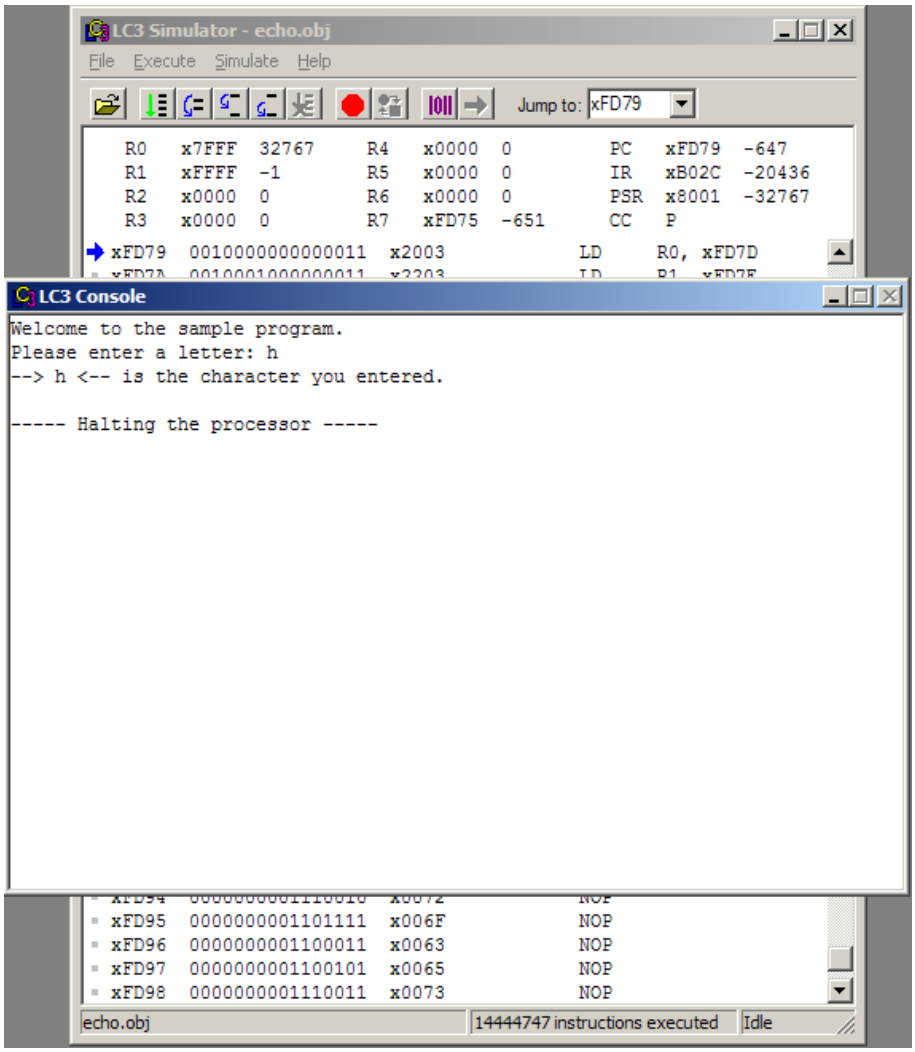

Figure 4: The sample file (echo.asm) has been run in the simulator.

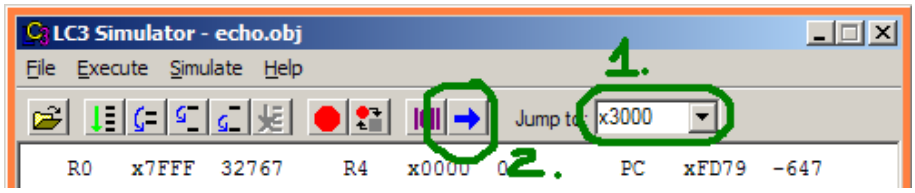

Figure 5: Jump to the starting address (defined by .orig – usually 0x3000 – in the program), and set the PC to that value, to run the program again.

deadline. Since this lab is due on Friday at 5pm, you will need to do this in your Wednesday or Thursday lab section.

If you end up needing a late check-off, we will do this (in person) using the files you submit in eCommons. You are only charged late days based on the submission time of these files and can

get checked off in your next lab section or in TA office hours (preferred).

Grading template

This is a suggested grading rubric. It is also a good general guideline before submitting your lab to check off these points.

☐ (5 pts) Does your program read from x3100 and write to x3101?

☐ (6 pts) Does your program iterate through the correct number of words?

☐ (7 pts) Does your lab3.bin divide the machine code into fields (opcode, registers, etc.) with spaces for readibility?

☐ (10 pts) Did you comment what each field of each instruction is?

☐ (12 pts) Did you comment the general methodology of your algorithm? (In lay terms, how does it work?)

☐ (15 pts) Does the program work?

Lab write-up requirements

In the lab write-up, we will be looking for the following things. The lab report is worth 20 points. We do not break down the point values; instead, we will assess the lab report as a whole while looking for the following content in the report.

- How many cycles does your program take to count the bits in a single word? Does this depend on the bit pattern?
- How many unique opcodes did you need?
- What type of branch/jump instruction did you use and how did you calculate the address/offset?
- How much memory (in bytes) does your program take?
- How many registers did your program need to use?
- What happens if you need more than 8 registers?
- What happens if you don't reset the registers that you use in a program?
- What additional instruction would allow your program to be more efficient?
- By stepping through the hello.bin program, what does the PUTS trap do?
- By stepping through the hello.bin program, what does the HALT trap do?