

MAT 3670: Lab 8

Introduction to the LC-3 Computer

Background

This lab introduces ideas related to the von Neumann¹ model and the LC-3 instruction set. The information found in Chapter 5 and summarized in Appendix A will be needed to complete this lab.

Our first experience programming the LC-3 computer is going to be at a low level, using **machine instructions**. Each instruction of the LC-3 is a 16-bit quantity — typically expressed as a 4-digit hexadecimal value or as a 16-digit binary value. A complete program is simply a sequence of 16-bit values. As you can imagine, while this is convenient for the hardware, it is much less so for people — we need to be especially careful, since making an error involving a single bit is easy to do.

Pre-Lab Exercises

1. A very small program for the LC-3 is shown in Figure 1. This program takes up just nine words of memory, beginning at `x3000`. In order to understand what each instruction of this program will do, we need to first view each 4-digit hexadecimal value as a 16-bit quantity, then consult Appendix A to determine the operation, operands, and instruction semantics. This has been done for a few instructions. Check these to test your understanding, then complete the table for the remaining entries.
2. Repeat Exercise 1 for a second LC-3 program, given in Figure 2.
3. If you want to get a head start on the lab, record your answers from the two tables in a `README` file. You only need to include the first and last columns from each table.
4. With pencil and paper, trace the execution of the second program. What will the values of registers `R0` through `R5` be when the computer reaches the `TRAP` instruction?
5. Figure 3 gives the skeleton of an LC-3 machine language program. Each line is missing 16 bits, which you are to supply. Using Appendix A as a reference, fill in the missing bits needed for each instruction. *Caution is needed here, as it is easy to make a mistake!* The last three words, beginning at address `x300B`, are not instructions — they are data words (i.e., variables).
6. Using Figure 4 and the LC-3 documentation, trace the execution of this program. This program makes use of two data values, found at `x300b` and `x300c`. In a few sentences, describe what this program is trying to accomplish. Predict the results of running this program.
7. Suppose the values in memory locations `x300b` and `x300c` are interchanged. With this change, what would the program do?
8. One of the lab exercises asks you to write an LC-3 program which will count the number of one bits in a specified word of memory. Think about how you might do this. A key question to ask is this: how can we inspect the *i*th bit to determine if it is a one?

¹Named after John von Neumann (1903–1957): Hungarian-American mathematician and pioneer of the digital computer.

Address	Content	Content (binary)	Instruction details
x3000	xE3FD	1110 001 1 1111 1101	LEA DR = R1 offset = x1FD R1 = PC + SEXT(x1FD) setcc()
x3001	x146E	0001 010 001 1 01110	ADD DR = R2 SR = R1 imm = 1 imm5 = x0E R2 = R1 + SEXT(x0E) setcc()
x3002	x35FB		
x3003	x54A0		
x3004	x14A5		
x3005	x744E		
x3006	xA7F7		
x3007	x300A		
x3008	xF025	1111 0000 0010 0101	TRAP trapvect8 = x25 halt execution

Figure 1: A sequence of machine instructions for the LC-3 computer. Compare with lab8a.asm.

Address	Content	Content (binary)	Instruction details
x3000	x5020	0101 000 000 1 0 0000	AND DR = R0 SR = R0 imm = 1 imm5 = x00 R0 = R0 AND SEXT(x00) setcc()
x3001	x1221		
x3002	xE404		
x3003	x6681		
x3004	x1262		
x3005	x16FF		
x3006	x03FD		
x3007	xF025	1111 0000 0010 0101	TRAP trapvect8 = x25 halt execution
x3008	x0006		data word

Figure 2: A sequence of machine instructions for the LC-3 computer. Compare with lab8b.asm.

Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	operation/data
x3000																	R1 = PC + 10
x3001																	R2 = M[R1 + 0]
x3002																	R3 = M[R1 + 1]
x3003																	R4 = NOT(R3)
x3004																	R4 = R4 + 1
x3005																	R5 = R2 + R4
x3006																	BRzp x3009
x3007																	M[R1+2] = R3
x3008																	BRnzp x300A
x3009																	M[R1+2] = R2
x300A																	TRAP x25
x300B																	x0036
x300C																	x0070
x300D																	xFFFF

Figure 3: The basic ingredients of an LC-3 program. Consult the documentation for the LC-3 ISA to fill in each of the missing bits.

PC	operation	R0	R1	R2	R3	R4	R5	R6	R7	n	z	p	M[x300B]
x3000	R1 = PC + 10												
x3001													

Figure 4: Predicting the execution of an LC-3 program.

Lab Exercises

1. Create an **lc3** folder to store the files needed for this lab. Obtain the files for this lab from the course web site, placing them in this folder. Here is a brief description of these files:

<code>PennSim.jar</code>	LC-3 simulator
<code>lc3os.asm</code>	LC-3 operating system
<code>lab8a.asm</code>	a first LC-3 program
<code>lab8b.asm</code>	a second LC-3 program

2. Launch the LC-3 simulator by double clicking on `PennSim.jar`. Enlarge the window by dragging the lower-right corner, which will give you a larger output window within the simulator.
3. Within the simulator, immediately above the output window, there is a place to enter a command. To get a summary of all the commands, click on this area, then enter `help`.
4. To run a program on the LC-3, we need to have one or more **object** programs. To produce these, we use the simulator's built-in assembler. For example, we can assemble the LC-3 operating system by giving the following command:

```
as -warn lc3os.asm
```

If all goes well, you will get a message indicating no errors or warnings were produced. Equally important, you will find a file named `lc3os.obj` was deposited in your working directory. In a similar manner, assemble the code found in `lab8a.asm`.

5. You can now load both of these object files into memory, using the following commands:

```
load lc3os.obj
load lab8a.obj
```

The simulator will provide feedback for each of these requests. Scroll through the LC-3 memory, locating the word at address `x3000`. Observe that its content is `xE3FD`, the first instruction of the `lab8a` program.

6. Observe that the value of the PC register is `x0200`, the required **entry point** for every LC-3 program we run. Also note that our own programs always have an “origin” of `x3000` — the first instruction of any program you write will always reside in memory at address `x3000`.
7. Set a **breakpoint** at the first instruction of our program with the following command:

```
break set x3000
```

8. Click on the **Continue** button to start execution of the LC-3. The memory word with address `x3000` will be highlighted in yellow. Now, carefully step through the program one instruction at a time by clicking on the **Step** button. After each instruction, notice how the registers change. Using the completed table from Figure 1, verify the actions taken by each instruction of the program.
9. Perform similar actions for the program in `lab8b.asm`. You will need to assemble `lab8b.asm`, then load `lab8b.obj`. To restart execution, change the value of the PC register to `x0200` then click on **Continue**. (To change the value of a register, double click on its value, enter the desired new value, then hit the enter key.)

In a `README` file, record the value of each of the registers R0 through R5 as a result of executing the program `lab8b`.

10. In the `README` file, record the first and last columns of your completed tables from Figures 1 and 2.
11. Using information you entered in Figure 3, use Aquamacs to create a file named `lab8c.asm`, following the pattern of our first two programs. Write each 16-bit instruction as a hexadecimal value and use `.FILL` statements.
12. Within `PennSim`, assemble and load your `lab8c` program, then step through it, instruction by instruction. Does it produce the result you predicted from your pre-lab exercises?
13. Swap the values of memory locations `x300b` and `x300c` and run the program again. Was your pre-lab prediction correct?
14. Design an LC-3 machine language program (i.e., essentially consisting of `.FILL` instructions) which will count the number of bits which are set in the word stored at the location which immediately follows the last instruction (i.e., `xf025`) of your program. At the conclusion of your program, the bit count (a number between 0 and 16) should be stored in `R0`.

Place your program in the file `lab8d.asm`. Thoroughly test your program. *Include explanatory comments within your program.*

Submissions

The submission process is slightly different from previous labs. When everything is working to your satisfaction, do the following:

- Create a `lab8` folder
- Make copies of the following files, placing them in the `lab8` folder:

```
README lab8c.asm lab8c.obj lab8d.asm lab8d.obj
```

- Now, submit your `lab8` folder in the usual way by dropping it on the EIU submit icon.

Appendix

Contents of lab8a.asm

```
1      ;;
2      ;; Author: Bill Slough
3      ;;
4      ;; MAT 3670
5      ;;
6      ;; A first machine-language LC-3 program
7      ;;
8      ;; This program is intentionally lacking comments, since part of the goal
9      ;; for this lab is to investigate the instruction set of the LC-3 computer
10     ;;
11
12     .ORIG x3000          ; specify the "origin"; i.e., where to load in memory
13
14     ; machine instructions
15     .FILL xE3FD
16     .FILL x146E
17     .FILL x35FB
18     .FILL x54A0
19     .FILL x14A5
20     .FILL x744E
21     .FILL xA7F7
22     .FILL x300A
23     .FILL xF025
24
25     .END
```

Contents of lab8b.asm

```
1      ;;
2      ;; Author: Bill Slough
3      ;;
4      ;; MAT 3670
5      ;;
6      ;; A second machine-language LC-3 program
7      ;;
8      ;; This program is intentionally lacking comments, since part of the goal
9      ;; for this lab is to investigate the instruction set of the LC-3 computer
10     ;;
11
12     .ORIG X3000        ; specify the "origin"; i.e., where to load in memory
13
14     ; machine instructions + data
15     .FILL x5020
16     .FILL x1221
17     .FILL xE404
18     .FILL x6681
19     .FILL x1262
20     .FILL x16FF
21     .FILL x03FD
22     .FILL xF025
23     .FILL x0006        ; data word
24
25     .END
```