CSCI 162 Lab 4

Purpose: To create a simple assembler language program.

Optional Resources: MARIE Download (This is already available on the school computers. You do not need to download it).

http://computerscience.jbpub.com/ecoa/2e/instructor_resources.cfm

Objectives: After this lab you should be able to

- Edit and run a simple MARIE program.

Background:

In this course we use a simple, hypothetical machine architecture called MARIE. This architecture is so simple, in fact, that no real machine would ever use it. For one thing, the continual need to fetch operands from memory would make the system very slow. Real systems minimize memory fetches by providing a sufficient number of addressable on-chip registers. Furthermore, the instruction set architecture of any real system would be much richer than MARIE's is.

Despite these limitations, there are many widely used concepts that can be learned by studying MARIE. Virtually every assembler in use today passes twice through the source code. The first pass assembles as much code as it can, while building a symbol table; the second pass completes the binary instructions using address values retrieved from the symbol table built during the first pass.

It is also useful to discover what a compiler is doing with the statements from a high level language as it converts it into binary, since there is a one-to-one correspondence between assembler and binary. Begin with a program statement in some high–level language.

Z = X + Y

In the MARIE assembly language, this would be written as follows.

```
Load    X
Add     Y
Store   Z
```

The hexadecimal representation of the MARIE machine language might be as follows.

```
10A2
30BC
202D
```

The full program might look like this:

```
Load X
Add Y
Store Z
Halt
X,    Dec  5 /The first number to add
Y,    Dec  7 /The second number we add
Z,    Dec  0 / The result
```

Here is the Instruction Set for MARIE

Instruction Number		Instruction	Meaning
Bin	Hex		
0001	1	Load *X*	Load the contents of address *X* into AC.
0010	2	Store *X*	Store the contents of AC at address *X*.
0011	3	Add *X*	Add the contents of address *X* to AC and store the result in AC.
0100	4	Subt *X*	Subtract the contents of address *X* from AC and store the result in AC.
0101	5	Input	Input a value from the keyboard into AC.
0110	6	Output	Output the value in AC to the display.
0111	7	Halt	Terminate the program.
1000	8	Skipcond	Skip the next instruction on condition.
1001	9	Jump *X*	Load the value of *X* into PC.

Instructions:

Go through each of these steps and answer the ***italicized*** questions on Moodle.

1. Create a directory called marie and cd into it.

2. `jar xvf  /home/faculty/beestonj/csci162/marie/MarieSim.jar`
3. `java MarieSim1`

4. Edit a new file using the **File -> Edit** menu option. Write an assembler program to add two numbers in memory whose values are 5 and 7 and store the answer in memory. Save it as **lab4.mas**. Assemble it.
   Check your directory -- what new files have been created?

5. Step through your program.
   Load the lab4.mex file: File ->Load
   Then press the [Step] button.
   How to the values of AC change at each step?
   What does PC represent?
   What values does MAR take on, and when does it change?

6. ***What do you think will happen if you have no HALT instruction at the end of the program?***
   Somebody in the lab should try it and report what happens.

7. Alter your program so it adds 2 to a value, originally zero, forever until you click on STOP. You can label lines of code, just like you label memory locations, and jump to the labelled line of code. Output your result at each iteration. Change the output format to Decimal to see the output. Save this file as JB123456789.mas where JB is your initials and 123456789 is your student number. Hand in this code via Moodle.

8. Add the line, ORG 100 to the beginning of your program.
   What is different when you assemble and run this code?

What to hand in:

1. Your answers to the questions.
2. Your code from question 7.