FIT9137 Applied Session

Week 2

Topics:

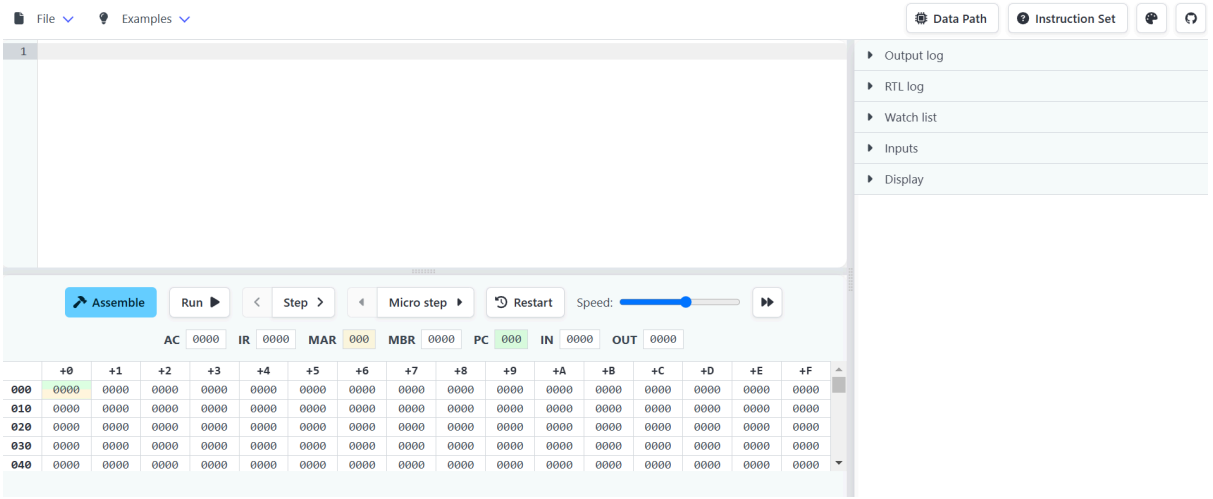
- The purpose of this applied session is getting to know MARIE computer architecture and assembly language.

Instructions:

- Please go through the pre-class (own-time) material before attending to these tasks.
- One of the main purposes of an applied session is to use the MARIE simulator to explore computer architecture and write assembly language programs. The other goal is to give and receive feedback from your peers and or your tutors.
- Form groups of 2 students (peers) to work through the exercises. If you meet a problem, try to solve it by asking direct questions to your peers. If the issue was not solved within peers, ask your tutor. If you did not get a chance to solve the problem during your applied session with your peer or tutor, jump into one of many consultation hours and ask any of the tutors to help you. Please visit the "Teaching Team and Unit Resources" tile in the FIT9137 Moodle site.

1. Brief Introduction to MARIE

The MARIE.js is a simulator built in JavaScript simulating a very simple computer architecture known as Machine Architecture that is Really Intuitive and Easy (MARIE) developed at Purdue University. We will use the MARIE simulator to learn about basic computer architecture and assembly language programming.



MARIE simulator user interface has it all in front of the user. It displays the (i) Code Entry window at the top left, (ii) CPU-registers window at the middle, just below the control buttons

to assemble and run, (iii) Memory locations (displayed in row by row fashion) with the address of the first location shown at first column, (iv) a multi-purpose window at the right that can be used to view: program output, input, graphic display, RTL-log and Watch-List. At the memory display section, you can scroll down to view the last memory location contents. The speed button at the middle can be used to control program execution speed.

2. MARIE Basics

Use the MARIE simulator to try out the program given below and answer the following questions. Copy the codes in the simulator input window, assemble your program to check for any errors, and then run or step through the lines of your codes to observe the changes taking place in CPU registers and various memory locations.

Line No.	Code	Comments
1	Load 004	/ Load value from address 004 into AC
2	Add 004	/ Add value from address 004 to value in AC
3	Store 004	/ Store AC into address 004
4	Halt	/ Stop execution
5	DEC 3	/ This is address 004, initialise with value 3

(a) Can you locate your program In MARIE memory? What are the memory addresses of the first and the last line of your program?
(b) In what format is your program stored in MARIE memory? How is your data "DEC 3" stored?
(c) What is the first and the last address of MARIE memory?
(d) Using a variable to refer to "DEC 3", we can modify the line no. 5 as follows:

5	num, DEC 3	/num is a variable and num = 3

Rewrite the program using the variable "num" instead of using the memory address to refer to the data "DEC 3"

3. Branch Instructions in MARIE and use of labels

Extend your program from the previous Activity, by adding labels, such as:

```
begin,    Load   num
          Add    num
          ……..
          …….
```

(a) Further modify your program by adding jump instructions to repeat the data input from keyboard, modify, store in memory and display activities endlessly.

(b) Using “skipcond” instruction, exit the program when the user inputs data ‘0’.

4. What does a “a program stored in memory“ look like? MARIE Disassembly.

This activity relates to the machine code that is generated after you “Assemble” your code. Your task is to “disassemble” a MARIE program, i.e., to reconstruct the original program based on the memory contents. You will have to refer to the instruction set in the MARIE simulator to find the machine code (in HEX notation) for each MARIE command.

Here is your first task. This is a screenshot of a MARIE memory:

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
000	5000	2005	3005	6000	7000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
010	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
020	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
030	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Reconstruct the MARIE program by looking at each memory cell and extracting the instruction and address argument. You can verify that your program is correct by entering it into the MARIE simulator, assembling it and comparing the memory contents.

Here is a memory screenshot for your second task:

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
000	5000	200B	100B	8800	9009	6000	400A	200B	9002	7000	0001	0000	0000	0000	0000	0000
010	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
020	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
030	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

5. Branch Instructions in MARIE and use of labels (optional/homework)

A conditional statement allows the flow of a program to depend on data. In high-level programming languages, these are typically achieved using “if-then-else” such as the following (in pseudocode):

```
if-then-else
        if (X > Y)
                Display X
        else
                Display Y
```

Write a MARIE assembly language program implementing the conditional statement.