FIT9137 Workshop
Week 2

Topics:

- Computer Architecture and Assembly Language

Covered Learning Outcomes:

- Describe basic concepts of Computer Architecture.
- Understand basic computer CPU components and their inter-actions.

Instructions:

- One of the main targets of workshops is to anchor the learner into the session and create many opportunities to reinforce the learning in different ways – individually and in small groups. Sometimes we also teach key practical/theoretical concepts to you during these sessions.
- Form groups of 4-5 students to work through the exercises. If you meet a problem, try to solve it within your group by discussing it with your group members. If not resolved within the group, ask one of the support tutors to help you.
- You still have a question? Jump into one of many consultation hours run by our experienced tutors and seek help. Please visit the "Teaching Team and Unit Resources" tile in the FIT9137 Moodle site.

Activity A: Von Neumann to MARIE Architecture

The Von Neumann architecture (or model) is the design of a computer showing the basic computer components (i) CPU, (ii) Memory and (iii) Input/Output devices. These three components are interconnected using a bunch of wires known as bus. We need a bus to carry data bits and a bus to carry address bits.

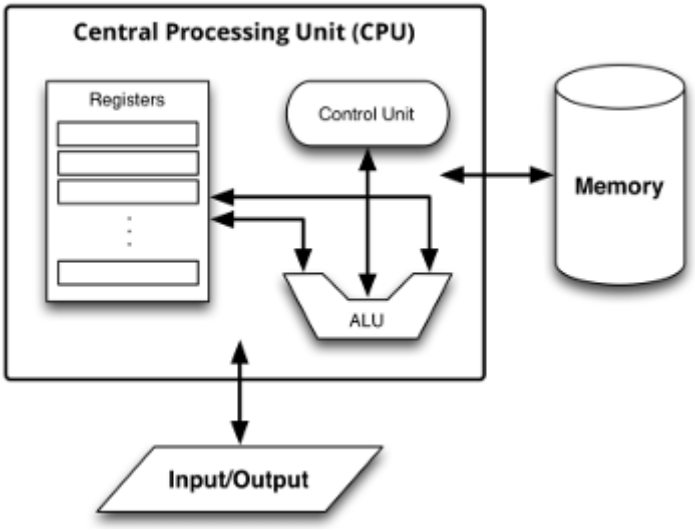

Figure 1: The Von Neumann Architecture (or Model)

We will use the MARIE simulator to explore different components of a computer. MARIE is a simple computer architecture designed at Purdue University having the basic computer components following the Von Neumann model. And, almost all modern computers are based on the Von Neumann architecture.

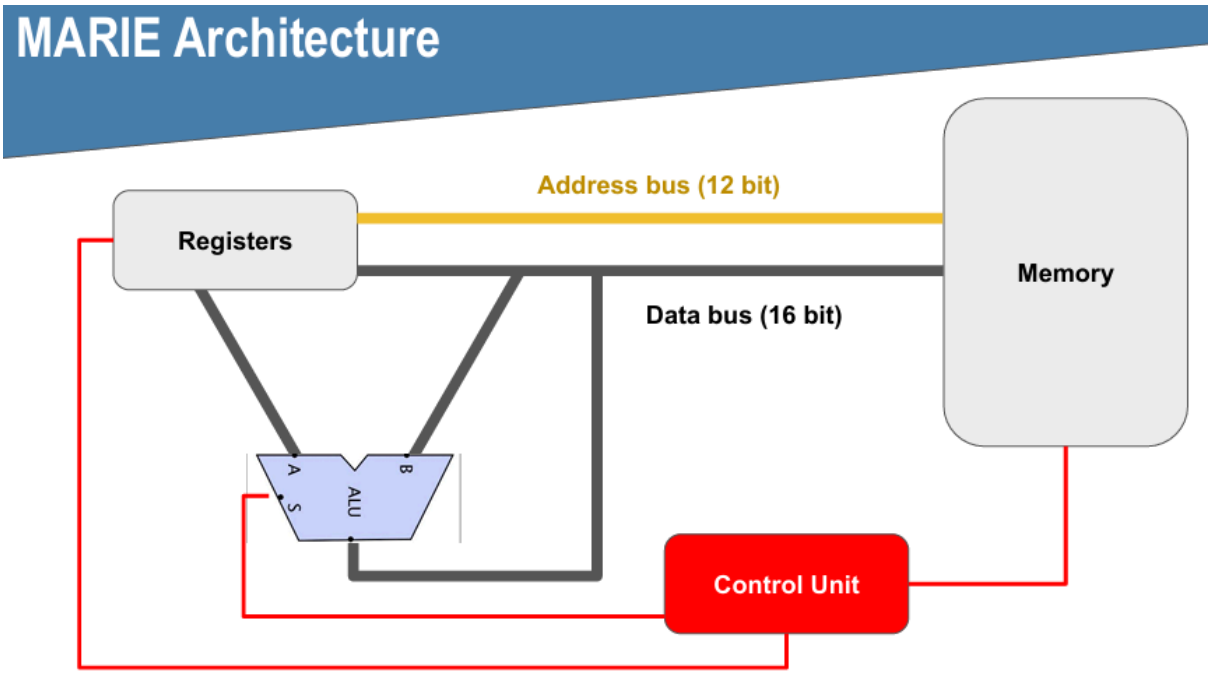

Figure 2: MARIE Architecture

Using the MARIE simulator explore different components (CPU, Memory and Address/Data Bus) and answer the following questions:

1. Name the components of the CPU. You can use the button "Data Path" to switch between viewing modes.
2. There are a number of registers in MARIE. How many are there? What is the number of bits each of them can hold?
3. In MARIE memory, we can store instructions and data of fixed lengths. What is that length? How many bits long instructions or data can we store in MARIE memory?
4. Explore the memory display in the simulator and find out how they are arranged for us to view the entire memory content. Any comments? Why is it arranged in 16 blocks (a word) in a row? How many memory locations are all together? Is there any relationship between this number and the length of the address bus, i.e.number bits in the address?
5. Any CPU needs a certain number of steps (micro-step) to execute any instruction. What is the maximum number of micro-steps MARIE can perform? Check the Control Unit step count for this.

Note: The instructions and data displayed in MARIE simulator registers and memory locations are in HEX notation. But they are stored in the memory in binary form. So, a memory content displayed as "004A" is stored in real memory hardware as "0000 0000 0100 1010". This is the default display mode for most of our computer displays apart from the other common format which is ASCII/Unicode.

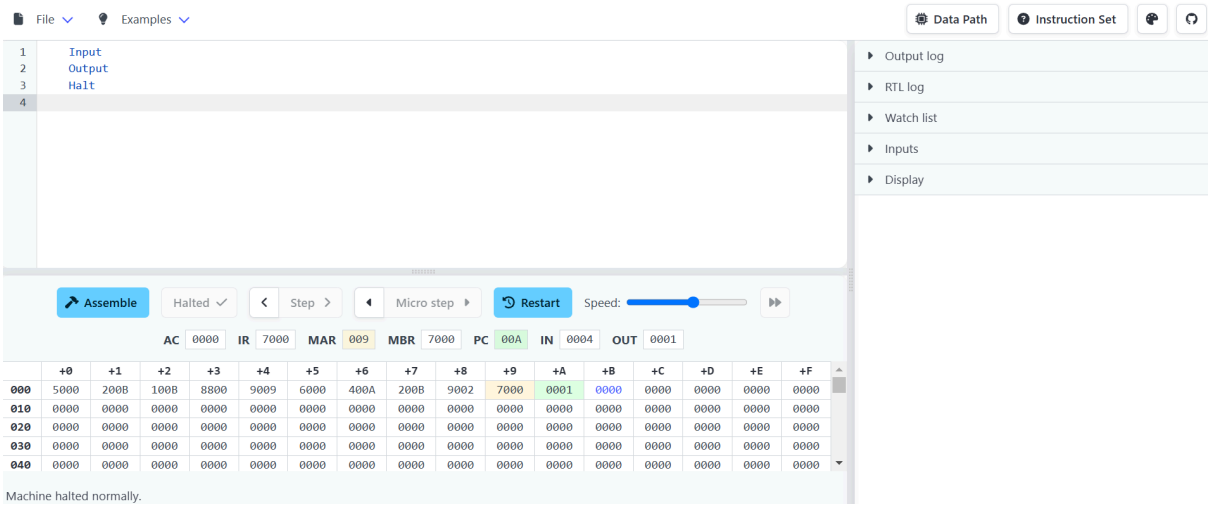

Figure 3: MARIE Simulator GUI

Activity B: Assembly to Machine Language in MARIE

A high-level computer program like the one below can't be executed (i.e handled) by any CPU. And these codes need to be compiled (and/or assembled, interpreted) to change them into machine code format.

```
import sys
name = sys.argv[1]
print 'Hello, ' + name + '!'
```

A CPU can only execute machine code which is written following its own format. So, machine code is different for each computer architecture. An example below is a bit pattern for a LOAD instruction in MARIE.

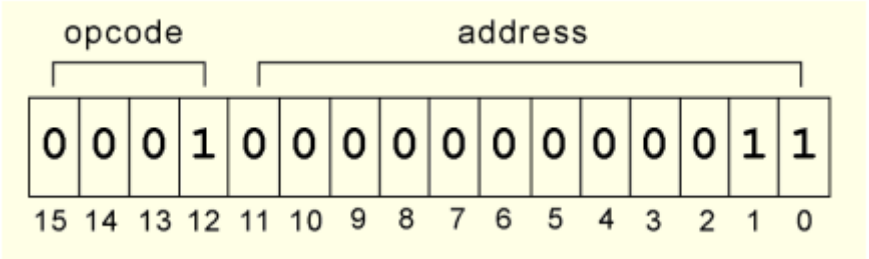

Figure 4: MARIE instruction set format

Machine code is difficult to read and write. Hence, we use assembly language to write programs and an assembler converts the assembly language program into machine code.

The corresponding assembly language code for the above machine code is: LOAD 003. We can see that the LOAD command has been changed (by the Assembler program) to the

opcode is “0001” and the address of the memory from where this instruction is copying the data is “003”. So, each machine code instruction has a mnemonic which is an easy to remember word (e.g. LOAD). The assembler translates assembly into machine code.

1. Can you find the Assembler (button) in the MARIE simulator? Also, locate the buttons to run or step through your assembly language program.
2. Is there any command to take data from KB or send data to display? Can we do mathematical operations in MARIE? How about multiplication and division?
3. When you write an assembly language program, how do you inform the CPU that you are at the end of your task? Do you think it is very important to inform the CPU?
4. Write a small program to input a number from KB and display it on the display window.

Activity C: A Simple MARIE Instruction

A computer’s instruction set architecture (ISA) specifies the format of its instructions and the basic operations that the computer can perform. The ISA is an interface between a computer’s hardware and its software. Some ISAs include hundreds of different instructions for processing data and controlling program execution.

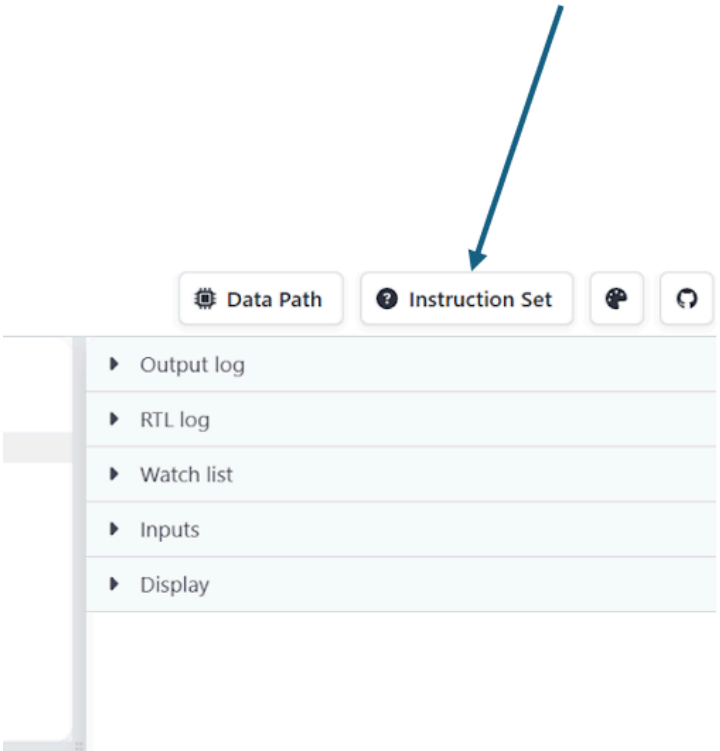

Figure 5: MARIE Instruction Set

1. How many instructions are there in the MARIE ISA? Can you name them? Do the names make any indication of their actions? You can explore a few simple instructions and their functional details.

2. Using the MARIE instruction set format shown in fig: 5, find out the opcodes for at least 6 MARIE instructions.

3. From the instructions format above, we can see that in an instruction, the address field is 12-bits long. What does it imply in computer design? Any relationship to the total memory addresses a MARIE computer can have?
4. Assume, you have a computer with opcode using 8 bits and the address field is 24 bits long. How many instructions are there in this computer's ISA? What are the total memory addresses this computer can have or can handle?

Activity D: The Control Unit in MARIE Architecture

In this activity we will explore the role of the Control Unit (CU) in instruction executions. Instruction execution involves data/instruction movement among different locations of the computer. We can see some attributes of the CPU and memory components in the figure below.

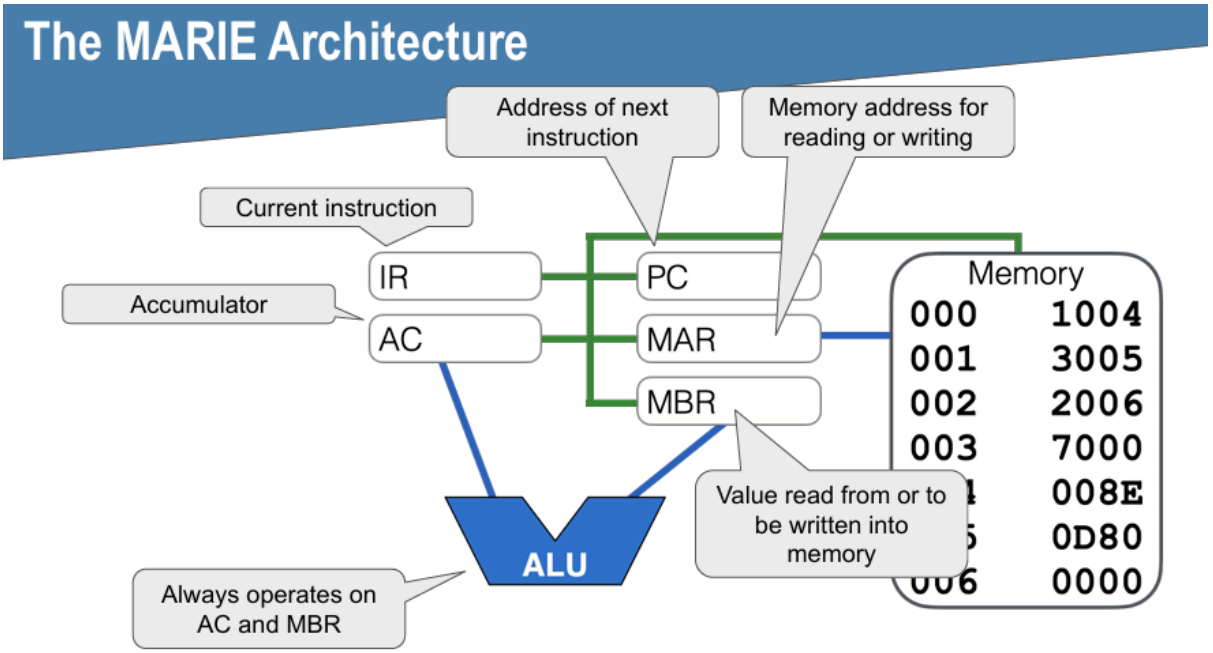

Figure 6: The MARIE Architecture

The CU also needs to coordinate registers, ALU and memory (based on your code). Each MARIE instruction actually consists of a sequence of smaller instructions called microoperations. The exact sequence of microoperations that are carried out by an instruction can be specified using register transfer language (RTL). The RTL for the instruction "Input" is shown below.

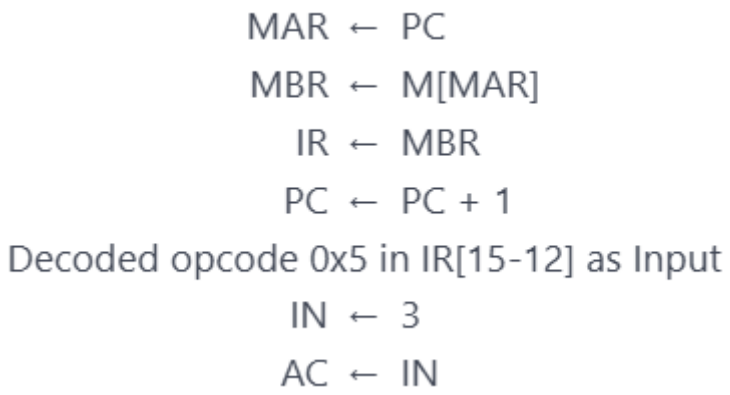

Figure 7: MARIE RTL (Register Transfer Language) for the command Input

In the MARIE RTL, we use the notation M[memory location] to indicate the actual data value stored in a memory location mentioned within the brackets [ ].

1. Use the MARIE simulator to find the RTL log window and investigate the micro-operations using the "micro-steps " button. You may use a simple command like "Input".
2. Can you identify the RTL steps for (i) Fetch the (next) instruction from memory, (ii) Decode the instruction, (iii) Execute the instruction from your RTL log?
3. Investigate RTL log for the following program:

```
//—----------------------
        Load myData
        Subt  one
        Store myData
        Halt
        myData, DEC 5
        One,    DEC 1
//—----------------------
```

4. Switch your MARIE simulator view to "Data Path" and observe the data movement for the "Input" command. Check the CU "step" lights changing from black to blue for every RTL movement. Each step counts one CPU clock pulse. Can you count the number of clock pulses (steps) needed for the "Input" command?
5. Check the number of steps/pulses for other commands as well. Are they the same? Do all the commands/instructions require the same amount of clock pulses?