# FIT9137 Applied Session

## Week 4

**Topics:**

- The purpose of this applied session is getting to know I/O in a computer system.

**Instructions:**

- One of the main purposes of an applied session is to use the MARIE simulator to understand how Input/Output works. 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. Data input, output and storing in memory (in MARIE)

Write a MARIE program to input any data from the keyboard and display in the MARIE simulator output window. Assemble your program to check for any errors, and then run or step through the lines of your codes and answer the following questions.

(a) How many different input and output modes of data are available in MARIE?

(b) In which register of the CPU is your input data stored?

(c) Add the following instruction "ORG 020" at the beginning of your program, and assemble your code again. Now, you will find that MARIE has relocated your machine code to a new address. What is this address? Is it the same as the contents of the CPU register PC? Why? Step through your program to see the changes happening in the PC.

(d) Add two variables to your program as follows:

```
numIn,      DEC 0
One,        DEC 1
```

Note: add these lines at the end of your program.

Modify your program to store the input data in "numIn" and add '1' to it before displaying it in the output. Please input decimal numbers (from the keyboard) for this task.

## 2. Input and Output: The journey from an I/O device to CPU-Registers

You must have noticed that in our MARIE working environment, we can input data from Keyboard and send out data to MARIE "Output log". Investigate both the data movements (i) from keyboard to the CPU registers and (ii) from the CPU registers to the output log. Name the devices and registers involved in the data movement process. You may use different view options (use Data Path) in the MARIE simulator environment to help you in the investigation.

## 3. Memory-mapped I/O: Graphics output in MARIE

Memory-mapped I/O means that we communicate with a device by reading from or writing to regular memory addresses. In the MARIE simulator, click on the "Display" tab (next to "Inputs") to see a display with 16x16 pixels.

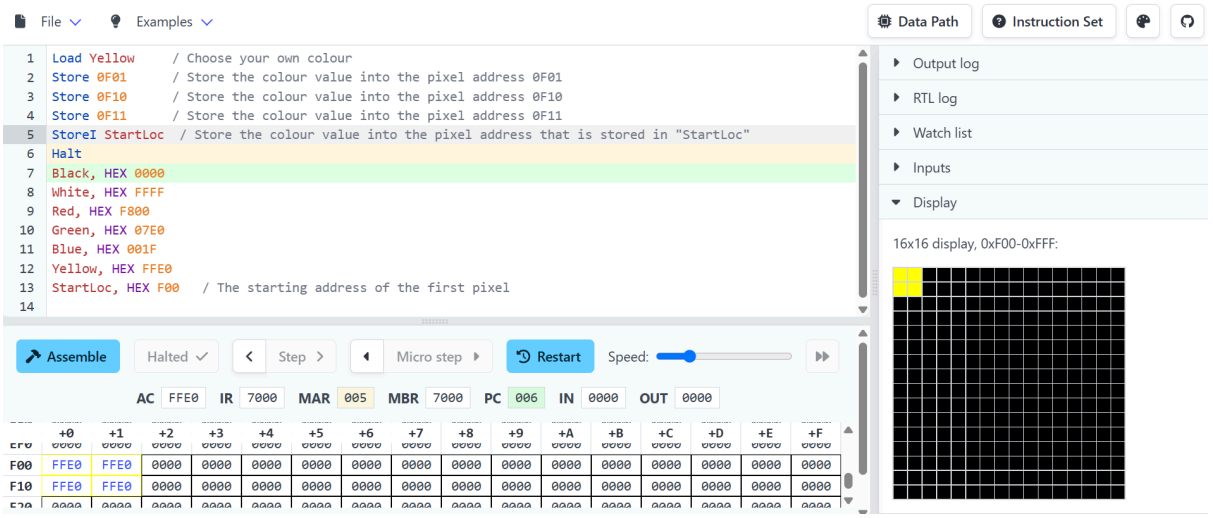


Figure 1: MARIE Simulator with Display window in view

Each pixel is mapped to a MARIE memory location in the range 0xF00-0xFFF. Writing to these memory locations will change the colour of the pixel (0 is black, FFFF is white, etc.).

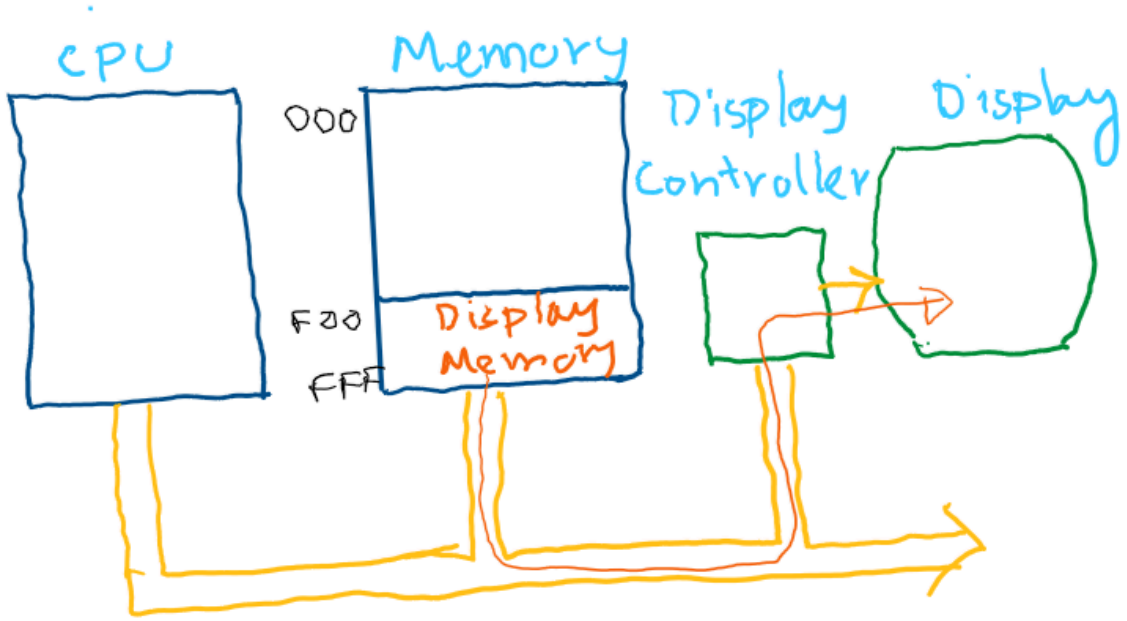


Figure 2: Display memory, display unit and the display controller

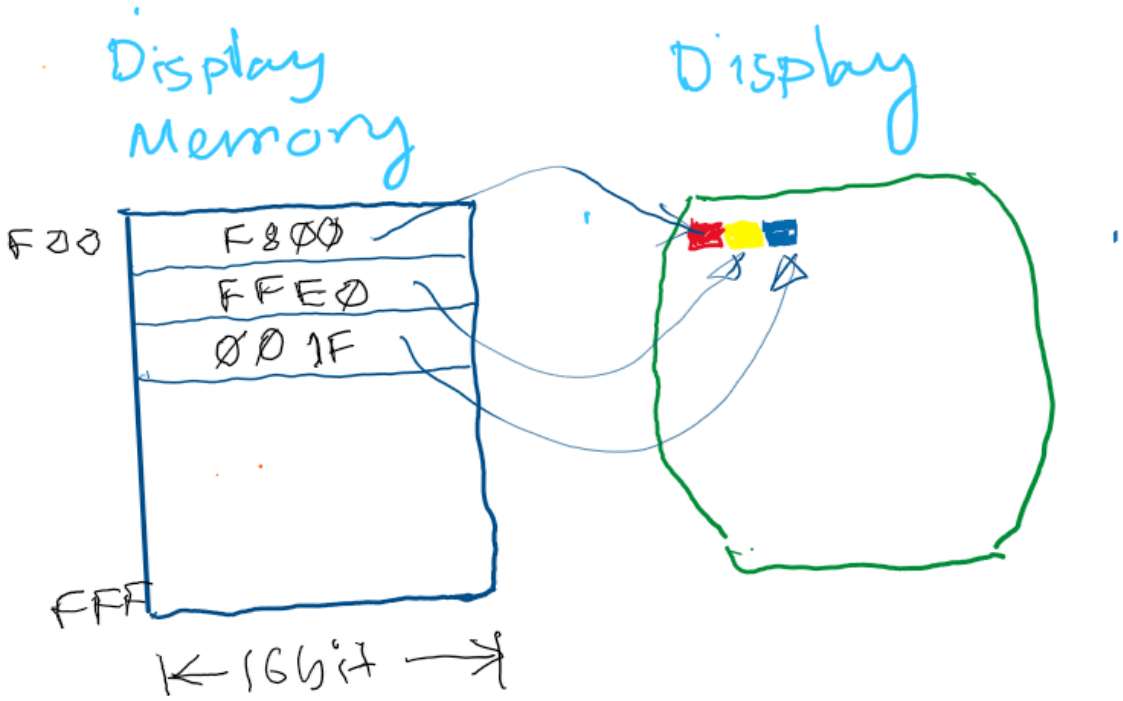


Figure 3: MARIE memory locations linking with the display window

(a) Write a MARIE program that draws something (colouring the pixels one after the other) on the graphics output display. You can use any of the two following templates to begin with.

(i) Template 1:
This template uses “Store” instruction to store the HEX value of a color in any location within the range of (0xF00-0xFFF) display memory addresses. You can repeat the instructions to create any pattern of your choice. Please note that in the second instruction, we write the address as “0F00” not as “F00” for the MARIE assembler to accept this as a HEX number ( a memory reference for location F00).

```
/-------------------------------------------------------------------------------
Load Yellow          / Choose your own colour
Store 0F00           / Store the colour value into the pixel address F00
Halt
/----color code HEX values-----
Black, HEX 0000
White, HEX FFFF
Red, HEX F800
Green, HEX 07E0
Blue, HEX 001F
Yellow, HEX FFE0
/-------------------------------------------------------------------------------
```

(ii) Template 2 (optional):
This template uses “StoreI” instruction to store the HEX value of a color in any location within the range of (0xF00-0xFFF) display memory addresses. This command (StoreI) will take the content of “startLoc” which is “HEX F00” as a memory address to store the data (from AC). You can repeat the instructions to create any pattern of your choice. By using jump instruction you can form loops to have a good program.

```
/—---------
Load Yellow       / Choose your own colour
StoreI startLoc   / Store colour value into pixel address stored in startLoc
Halt
/----color code HEX values-----
Black, HEX 0000
White, HEX FFFF
Red, HEX F800
Green, HEX 07E0
Blue, HEX 001F
Yellow, HEX FFE0
startLoc, HEX F00   / The starting address of the first pixel
/—-----------
```

Note: The MARIE command “Store” directly saves the value from the accumulator into a memory address mentioned in the instruction.

| Standard Format | Store [MemoryReference-direct] |
|---|---|
| Example 1:<br>Input data will be stored in memory location “009” | Input<br>Store 009<br>Halt |
| Example 2:<br>Input data will be stored in num1. Here, num1 is a variable with an initial value ‘0’ stored. In this case we don’t know which memory location the input data is stored. We only know that num1 is stored in a variable. | Input<br>Store num1<br>Halt<br>num1, DEC 0 |

The MARIE command “StoreI (Store Indirect)” on the other hand, saves the value from the accumulator into a memory location that is found in the variable mentioned in the instruction (as a memory reference). This instruction first retrieves an address from memory (using the memory reference given) and then stores the value at that retrieved address.

| Standard Format | StoreI [MemRef-Indirect] |
|---|---|
| Example 1 :<br>Input data will be stored in memory location “100”, and this address is found in the variable “Location1 (=100)” | Input<br>StoreI  Location1<br>Halt<br>Location1,  HEX 100<br>Num1,  DEC 0 |

In short, “Store” command directly targets a memory location, while “StoreI” follows an indirect path by looking up the actual storage address from another memory location. Indirect addressing (StoreI) is out of the scope of this unit. However, it's quite common in assembly language programming.

(b) Write a MARIE program that draws a vertical line on the graphics output display where four pixels are red, the next four are blue etc. Can this line be drawn at any column of the display? Optional task: can the user decide on which column this line is to be drawn during run-time of the program? You may have to use Template 2 for this part of the task.