CSE100 Lecture03
Machines, Instructions, and Programs
Introduction to Computer Systems

M.A.Hakim Newton

Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka – 1000, Bangladesh

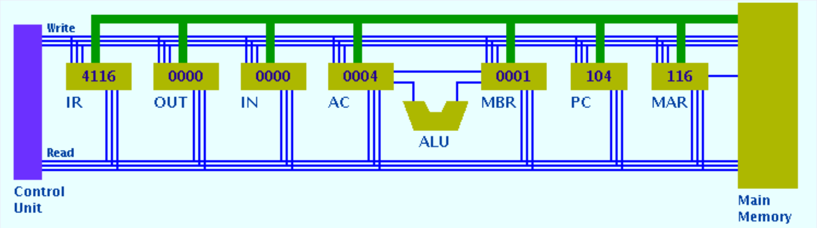
CSE, BUET, 2009

MARIE: An Example Machine

MARIE Simulator

- A really-inituitive and easy-to-use simulated computer.
- It helps quickly understand how computers really work
- The architecture has various fundamental features.

MARIE Architecture



MARIE Inside Details

MainMemory holds data and program both. MARIE has 12-bit memory addresses meaning 2^{12} locations. Each location contains 16-bit data; all data are signed meaning the range is -32768 to +32767.

InstructionSet contains only 13 instructions. All instructions run in the memory operand mode. So each instruction has a 16-bit op-code – 4 bits for the instruction code and 12 bits for the memory address.

IntructionRegister (IR) internally holds the current instruction op-code being exectued by the computer.

ProgramCounter (PC) internally holds the memory address of the next instruction to be executed.

MemoryAddressRegister (MAR) internally holds the address of the memory location to be read from or written into.

MARIE Inside Details . . .

MemoryBufferRegister (MBR) internally holds the data value to be written into the memory or the data value read from the memory.

Accumulator (AC) holds the data value to be operated on. This register is available to the user during programming.

Input (IN) holds the data value to be read from an input port/device. This register is available to the user during programming.

Output (OUT) holds the data value to be written to an output port/device. This register is available to the user during programming.

ControlUnit is responsible for sending necessary (read/write) signals timely to the memory and the registers. The signals depend on the current instruction in the IR.

MARIE Instruction Set

Mnemonic	Hex	Description
Clear	A	Put all zeros in AC
Add X	3	Add the value of address X to AC
AddI X	B	Add indirect: Use the value at X as the address of the data operand to add to AC
Subt X	4	Subtract the value of address X from AC
Input	5	Input a value from the keyboard into AC
Output	6	Output the value in AC to the display
Load X	1	Load the value of address X into AC
Store X	2	Store the value of AC at address X
Halt	7	Terminate program

MARIE Instruction Set ...

Mnemonic	Hex	Description
Jump X	9	Load the value of X into PC
JumpI X	C	Use X's value as the address to jump to
JnS X	0	Store the PC at X and jump to X+1
Skipcond X	8	Skip next instruction on condition. Bits 10 and 11 of X specify the condition. If the two bits are 00, this translates to "skip if the AC is negative". If the two bits are 01, this means "skip if the AC is equal to 0". Finally, if the two bits are 10, this translates to "skip if the AC is greater than 0". For example, Skipcond 800 if AC > 0, Skipcond 400 if AC $= 0$, and Skipcond 000 if AC < 0

MARIE Assembly Program

```
        ORG 100        /Program will start at memory address 100
If,     Load X         /Load the first value
        Subt Y         /Subtract the value of Y, store result in AC
        Skipcond 400   /If AC=0, skip the next instruction
        Jump Else      /Jump to Else part if AC is not equal to 0
Then,   Load X         /Reload X so it can be doubled
        Add X          /Double X
        Store X        /Store the new value
        Jump Endif     /Skip over the false, or else, part to end of if
Else,   Load Y         /Start the else part by loading Y
        Subt X         /Subtract X from Y
        Store Y        /Store Y-X in Y
Endif,  Halt           /Terminate program (it doesn't do much!)
X,      Dec 12         /Value of the variable X
Y,      Dec 20         /Value of the variable Y
        END
```

MARIE Object/Executable Program

```
                  ORG 100        /Program will start at memory address 100
100  110C  If,    LOAD X         /Load the first value
101  410D         SUBT Y         /Subtract the value of Y, store result in AC
102  8400         SKIPCOND 400   /If AC=0, skip the next instruction
103  9108         JUMP Else      /Jump to Else part if AC is not equal to 0
104  110C  Then,  LOAD X         /Reload X so it can be doubled
105  310C         ADD X          /Double X
106  210C         STORE X        /Store the new value
107  910B         JUMP Endif     /Skip over the false, or else, part to end of if
108  110D  Else,  LOAD Y         /Start the else part by loading Y
109  410C         SUBT X         /Subtract X from Y
10A  210D         STORE Y        /Store Y-X in Y
10B  7000  Endif, HALT           /Terminate program (it doesn't do much!)
10C  000C  X      DEC 12         /Value of the variable X
10D  0014  Y      DEC 20         /Value of the variable Y
                  END
```

Programming for MARIE Machines

How to write programs?

- Write the assembly program using the mnemonics.
- Translate the assembly program into op-codes (numbers).

How to obtain a translator program?

- Initially there is no translator/assembler program..
- Your first program should be a small assembler.
- Translate the assembler program completely manually.
- This time write probably a large assembler program.
- Assemble the large assembler using the small assembler.

High Level Programming for MARIE

A High Level Program

$X = 5$
$Y = 7$
$Z = X + Y$

A Low Level Program

Load X	X, Dec 5
Add Y	Y, Dec 7
Store Z	Z, Dec 0

High Level to Executable Programs

- Write a high level program; which is easier than writing an assembly program. Currently no high level language exists.
- Using a compiler program, compile the high level program into an assembly program. Currently no compiler exists.
- Using an assembler program, assemble the assembly program into an executable program. Currently an assembler exists.

MARIE Program Execution

Program Loading for Execution?

- The executable program is copied to the memory at the memory location specified in the assembly program.
- Register PC is set with the first memory address of the program and the system clock starts running.

How instructions are executed by the Control Unit?

- The instruction at the memory location determined by the current value of the PC is fetched to IR.
- The value of the IR is then decoded and the meaning of the instruction is understood.

MARIE Program Execution . . .

How instructions are executed by the Control Unit? . . .

- The value of the PC is incremented so that it now holds the address where the next instruction will be fetched from.
- Depending on the instruction in the IR, other read and write signals are sent to the registers to have the desired result.
- When the current instruction execution is finished, the next cycle begins which fetch, decode, and execute the next instruction in the memory.
- This continues until the HALT instruction is executed.

Program Execution: Running vs Tracing

Running vs Tracing a Program

- Running means executing all instructions at one time.
- Tracing means executing only one instruction at a time.

Tracing or Single Stepping for Debugging)

- Assemble can detect only syntactical bugs or mistakes.
- For semantical bugs, we need to trace the program.
- An example semantical bug is Subt X instead of Add X.
- To catch this bug, we need to examine values of the registers and memory addresses after every instruction.