جامعة ألأنبار

كلية علوم الحاسوب وتكنولوجيا المعلومات

قسم أنظمة شبكات الحاسوب

المرحلة الثانيه

Computer Architecture

التدريسي: أ.م.د عمر منذر حسين

MARIE: An Introduction to a Simple Computer

1- Register Transfer Notation

- We have seen that digital systems consist of many components, including ALUs, registers, memory, decoders, and control units.
- These units are interconnected by buses to allow information to flow through the system.
- The instruction set presented for MARIE in the preceding sections constitutes a set of machine level instructions used by these
- components to execute a program.
- Each instruction appears to be very simplistic; however, if you examine what actually happens at the component level, each instruction involves **multiple operations**.
- For example, the Load instruction loads the contents of the given memory location into the AC register. But, if we observe what is happening at the component level, we see that multiple “mini-instructions ” are being executed.
- First, the address from the instruction must be loaded into the MAR.
- Then the data in memory at this location must be loaded into the MBR.
- Then the MBR must be loaded into the AC.
- These mini-instructions are called **micro operations** and specify the elementary operations that can be performed on data stored in registers.
- The symbolic notation used to describe the behavior of micro operations is called **register transfer notation (RTN) or register transfer language (RTL)**.
- We use the notation M[X] to indicate the actual data stored at location X in memory, and $\leftarrow$ to indicate a transfer of information.

- In reality, a transfer from one register to another always involves a transfer onto the bus from the source register, and then a transfer off the bus into the destination register.
- However, for the sake of clarity, we do not include these bus transfers, assuming that you understand that the bus must be used for data transfer.
- We now present the register transfer notation for each of the instructions in the ISA for MARIE.
- **Load X**
  This instruction loads the contents of memory location X into the **AC**. However, the address X must first be placed into the **MAR**. Then the data at location **M[MAR]** (or address X) is moved into the **MBR**. Finally, this data is placed in the **AC**.

  MAR← X

  MBR← M[MAR]

  AC← MBR
- Because the IR must use the bus to copy the value of X into the MAR, before the data at location X can be placed into the MBR, this operation requires two bus cycles.
- Therefore, these two operations are on separate lines to indicate they cannot occur during the same cycle.
- Because we have a special connection between the MBR and the AC, the transfer of the data from the MBR to the AC can occur immediately after the data is put into the MBR, without waiting for the bus.
- **Store X**
  This instruction stores the contents of the AC in memory location X:

MAR← X

MBR← AC

M [MAR] ← MBR

- **Add X**
  The data value stored at address X is added to the AC. This can be accomplished as follows:

  MAR← X

  MBR← M [MAR]

  AC← AC + MBR

- **Subt. X**
  Similar to Add, this instruction subtracts the value stored at address X from the accumulator and places the result back in the AC:

  MAR← X

  MBR← M [MAR]

  AC← AC – MBR

- **Input**
  Any input from the input device is first routed into the InREG. Then the data is transferred into the AC.

  AC← InREG

- **Output**
  This instruction causes the contents of the AC to be placed into the OutREG, where it is eventually sent to the output device.

  OutREG← AC

- **Halt**

No operations are performed on registers; the machine simply ceases execution.

- **Skipcond**
  Recall that this instruction uses the bits in positions 10 and 11 in the address field to determine what comparison to perform on the AC. Depending on this bit combination, the AC is checked to see whether it is negative, equal to zero, or greater than zero. If the given condition is true, then the next instruction is skipped. This is performed by incrementing the PC register by 1.

```
if IR[11–10] = 00 then {if bits 10 and 11 in the IR are both 0}
  If AC < 0 then PC ← PC+1
  else If IR[11–10] = 01 then {if bit 11 = 0 and bit 10 = 1}
  If AC = 0 then PC ← PC + 1
  else If IR[11–10] = 10 then {if bit 11 = 1 and bit 10 = 0}
  If AC > 0 then PC ← PC + 1
```

- **If the bits in positions ten and eleven are both ones, an error condition results.**
- However, an additional condition could also be defined using these bit values.
- **Jump X**
  This instruction causes an unconditional branch to the given address X. Therefore to execute this instruction, X must be loaded into the PC.

PC← X

In reality the lower or least significant 12 bits of the instruction register (or IR[11–0]) reflect the value of X. So this transfer is more accurately depicted as:

PC← IR [11–0]

- However, we feel that the notation PC ← X is easier to understand and relate to the actual instructions, so we use this instead.
- Register transfer notation is a symbolic means of expressing what is happening in the system when a given instruction is executing.
- RTN is sensitive to the data path, in that if multiple micro operations must share the bus, they must be executed in a sequential fashion, one following the other.

2- INSTRUCTION PROCESSING

- All computers follow a basic machine cycle: the fetch, decode, and execute cycle.

2.1 The Fetch–Decode–Execute Cycle

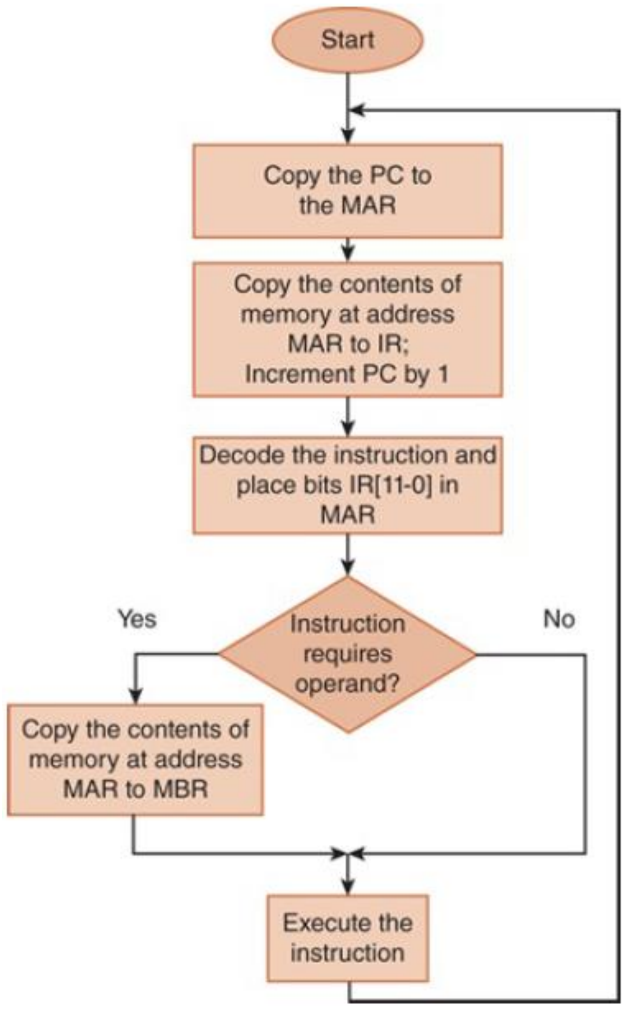
- The fetch–decode–execute cycle represents the steps that a computer follows to run a program.
- The CPU fetches an instruction (transfers it from main memory to the instruction register), decodes it (determines the opcode and fetches any data necessary to carry out the instruction), and executes it (performs the operation[s] indicated by the instruction).
- Notice that a large part of this cycle is spent copying data from one location to another.
- When a program is initially loaded, the address of the first instruction must be placed in the PC.
- The steps in **this cycle**, which take place in specific clock cycles, are listed below.
- Note that Steps 1 and 2 make up the fetch phase, Step 3 makes up the decode phase, and Step 4 is the execute phase.
  1. Copy the contents of the PC to the MAR: MAR ← PC.
  2. Go to main memory and fetch the instruction found at the address in the MAR, placing this instruction in the IR; increment PC by 1 (PC now

points to the next instruction in the program): IR ← M[MAR] and then PC ← PC + 1. (Note: Because MARIE is word addressable, the PC is incremented by 1, which results in the next word's address occupying the PC. If MARIE were byte addressable, the PC would need to be incremented by 2 to point to the address of the next instruction, because each instruction would require 2 bytes.)

3. Copy the rightmost 12 bits of the IR into the MAR; decode the leftmost 4 bits to determine the opcode, MAR ← IR[11-0], and decode IR[15–12].

4. If necessary, use the address in the MAR to go to memory to get data, placing the data in the MBR (and possibly the AC), and then execute the instruction MBR ← M[MAR] and execute the actual instruction.

- This cycle is illustrated in the flowchart in Figure 4.11.


Start
Copy the PC to the MAR
Copy the contents of memory at address MAR to IR; Increment PC by 1
Decode the instruction and place bits IR[11-0] in MAR
Yes
Instruction requires operand?
No
Copy the contents of memory at address MAR to MBR
Execute the instruction

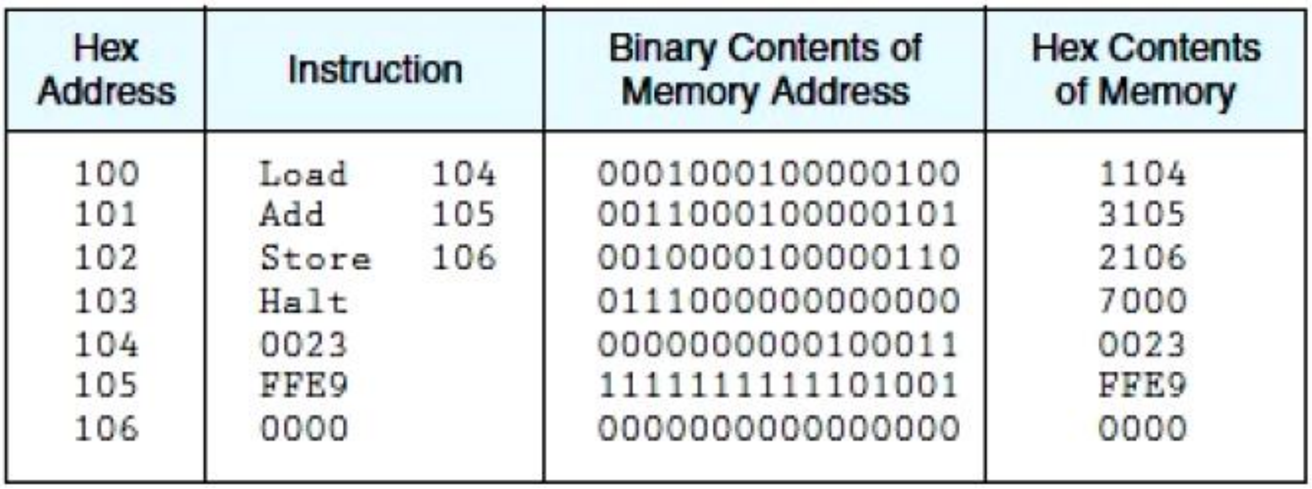
- Note that computers today, even with large instruction sets, long instructions, and huge memories, can execute millions of these fetch–decode–execute cycles in the blink of an eye.

2.2 Interrupts and the Instruction Cycle

- All computers provide a means for the normal fetch–decode–execute cycle to be interrupted.
- These interruptions may be necessary for many reasons, including a program error (such as division by 0, arithmetic overflow, stack overflow, or attempting to access a protected area of memory); a hardware error (such as a memory parity error or power failure); an I/O completion (which happens when a disk read is requested and the data transfer is complete); a user interrupt (such as hitting Ctrl-C or Ctrl-Break to stop a program); or an interrupt from a timer set by the operating system (such as is necessary when allocating virtual memory or performing certain bookkeeping functions).
- All of these have something in common: They interrupt the normal flow of the fetch–decode–execute cycle and tell the computer to stop what it is currently doing and go do something else. They are, naturally, called interrupts.

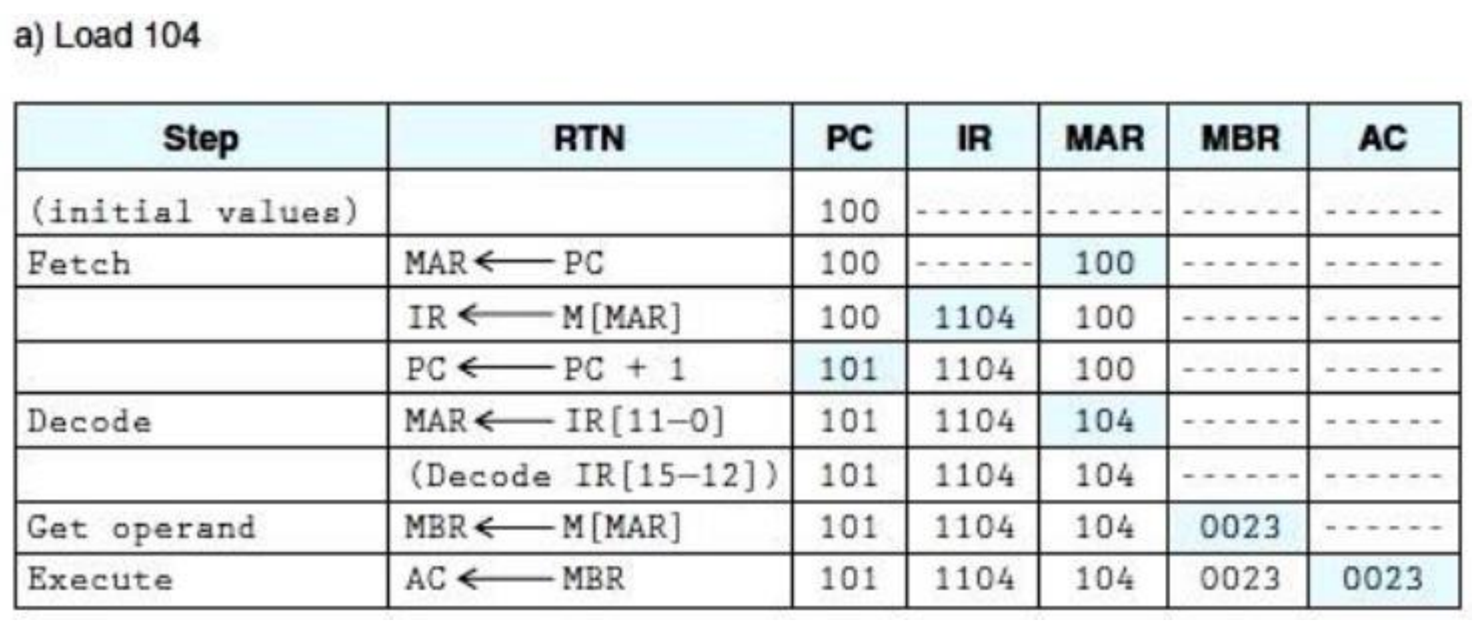
3- A SIMPLE PROGRAM

- Consider the simple MARIE program given below. We show a set of mnemonic instructions stored at addresses 100 - 106 (hex):



Hex Address	Instruction	Binary Contents of Memory Address	Hex Contents of Memory
100	Load 104	0001000100000100	1104
101	Add 105	0011000100000101	3105
102	Store 106	0010000100000110	2106
103	Halt	0111000000000000	7000
104	0023	0000000000100011	0023
105	FFE9	1111111111101001	FFE9
106	0000	0000000000000000	0000

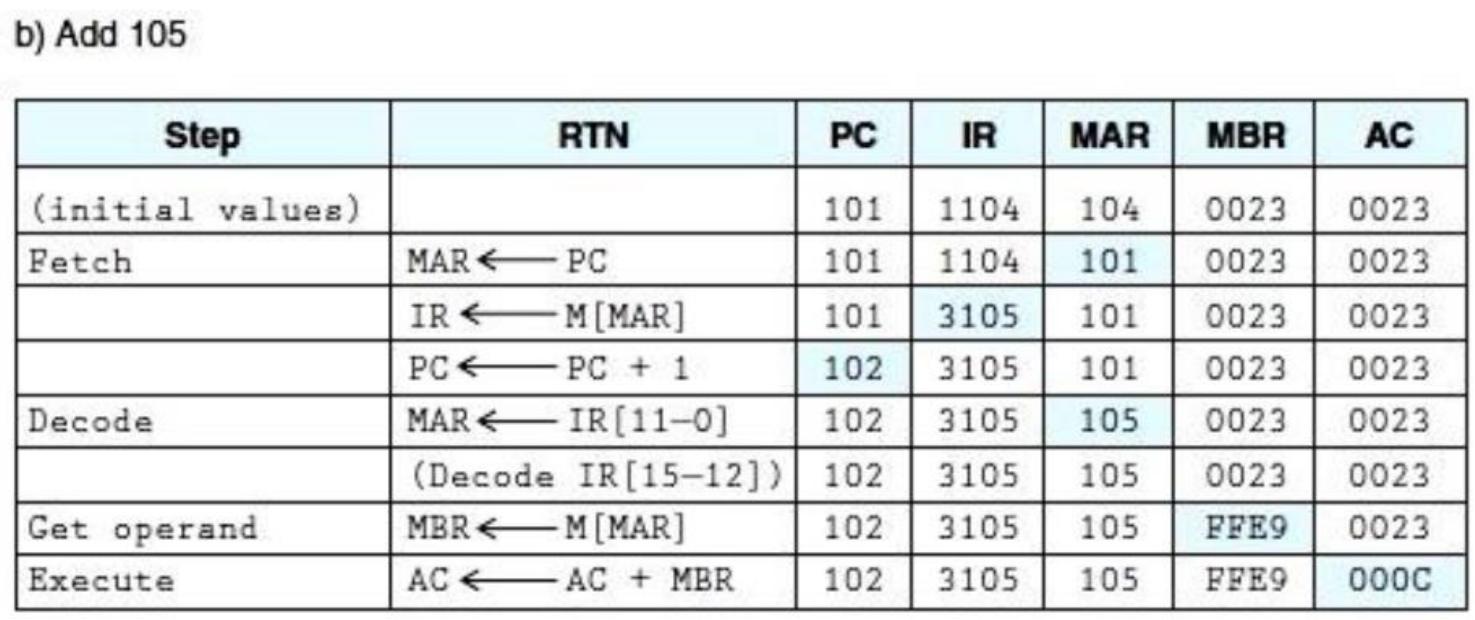
- Let's look at what happens inside the computer when our program runs.
- This is the **LOAD 104** instruction:



a) Load 104

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		100	------	------	------	------
Fetch	MAR ⟵ PC	100	------	100	------	------
	IR ⟵ M[MAR]	100	1104	100	------	------
	PC ⟵ PC + 1	101	1104	100	------	------
Decode	MAR ⟵ IR[11-0]	101	1104	104	------	------
	(Decode IR[15-12])	101	1104	104	------	------
Get operand	MBR ⟵ M[MAR]	101	1104	104	0023	------
Execute	AC ⟵ MBR	101	1104	104	0023	0023

- Our second instruction is **ADD 105**.



b) Add 105

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		101	1104	104	0023	0023
Fetch	MAR ⟵ PC	101	1104	101	0023	0023
	IR ⟵ M[MAR]	101	3105	101	0023	0023
	PC ⟵ PC + 1	102	3105	101	0023	0023
Decode	MAR ⟵ IR[11-0]	102	3105	105	0023	0023
	(Decode IR[15-12])	102	3105	105	0023	0023
Get operand	MBR ⟵ M[MAR]	102	3105	105	FFE9	0023
Execute	AC ⟵ AC + MBR	102	3105	105	FFE9	000C

c) Store 106

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		102	3105	105	FFE9	000C
Fetch	MAR ⟵ PC	102	3105	102	FFE9	000C
	IR ⟵ M[MAR]	102	2106	102	FFE9	000C
	PC ⟵ PC + 1	103	2106	102	FFE9	000C
Decode	MAR ⟵ IR[11–0]	103	2106	106	FFE9	000C
	(Decode IR[15–12])	103	2106	106	FFE9	000C
Get operand	(not necessary)	103	2106	106	FFE9	000C
Execute	MBR ⟵ AC	103	2106	106	000C	000C
	M[MAR] ⟵ MBR	103	2106	106	000C	000C