Microprocessors and Micro-controllers

Class 6

Classification of Instruction based on number of operand address they contain:

(i) **0 – Address Instruction**

Do not contain any operand address. Operand address is expressed implicitly which is the Accumulator. Monadic operations like Shift and Compliment are 0-Address instructions.

(ii) **1 – Address Instruction**

Only one operand address is specified in the instruction, which may be a register name or a memory address. The other operand address is implied, which is present in the opcode itself.

(iii) **2 - Address Instruction**

Here both the operand address is specified. The result is place in one of the specified address.

(iv) **3 – Address Instruction**

Two addresses are specified for two operand and one address for result.

Addressing Modes

- Each instruction needs data on which it has to perform specified operation.
- This operand (data) may be in the
  - Accumulator
  - General Purpose Registers
  - In some specified location in the memory.
- The techniques of specifying the address of the data are known as addressing modes.
- The important Addressing Modes are:

(i) Direct or Absolute Addressing:

- The address of the data is specified in the instruction itself.
- e.g. STA 2500H → (Stores the contents of the Accumulator in 2500H)
  LDA 2500H → (Loads A with the contents of 2500H)

(ii) Register Addressing:

- The operands are located in the registers (GPRs), i.e. the content of the register is the operand.
- e.g. MOV A, B (Transfers the contents of B to register A)

(The opcode for this instruction is 78H. In addition to the operation to be performed, the OPCODE also specified the address of the register mentioned in the Instruction.

78H, in Binary is 0111 1000, where 01→ denotes MOV, 111→ is the binary for register A, 000→ Binary for register B).

- ADD B (Adds the contents of register B to the contents of the Accumulator).

(The opcode for this instruction is 80H. The binary code is 1000 0000. The first 5 bits 10000→ specifies the operation to be performed. Last three bits 000 → is the binary code for register B).

(iii) Register Indirect Addressing:

- Here the address of the operand is given indirectly. The contents of the register or the register pair are the address of the operand.
- LXI H, 2400H (Load HL pair with 2400H. Move the content of the memory location whose address is in HL pair (2400H) to the accumulator)
- MOV A, M (In this case, the address of the memory location is not directly given in the instruction. The address of the memory location is stored in HL Pair which has been specified by earlier instruction "LXI H, 2400H").
- LXI H, 2200H (Loads HL Pair with 2200H)
- ADD M (Adds the contents whose address is in HL pair to the Accumulator).

(iv) Immediate Addressing

The operand is given in the instruction itself.

- MVI A, 06 (Move 06 to Accumulator)
- ADI 05 (Add 05 to the contents of A)
- LXI H, 2500H (Load HL Pair with 2500H)

(v) Implicit (or Implied) Addressing

These instructions operate only on one operand, which is the Accumulator.

- RAL (Rotate the contents of Accumulator left).
- RLC (Rotate the contents of Accumulator left through carry)
- CMA (take compliment of the content of A).

Some other addressing modes are:

vi. Indexed Addressing
vii. Based Addressing
viii. Based Index Addressing
ix. Relative Addressing
x. Relative Indexed Addressing

xi. Page Addressing

xii. Stack Addressing

Intel 8085 uses addressing modes only from (i) to (v).

Definitions of certain terms related to Addressing Modes:

- A large memory is divided into segments.
- The memory address of an operand (data) consists of two components:
    - The starting address of the segment and
    - An offset within the segment
- The starting address of the segment is supplied by the processor.
- The offset is determined by adding any combination of three offset address elements:
    - Displacement
    - Base
    - Index
- The combination depends on the addressing mode of the instruction to be executed.
- The offset is also called effective address.
- **The memory address of an operand = Starting address of the segment + offset**
-

Displacement	: A 8-bit or 16-bit immediate value given in the instruction
Base	: Contents of the Base register
Index	: Content of the index register

(vi) Indexed Addressing

The operand's offset is determined by adding an 8-bit or 16-bit displacement (given in the instruction) to the contents of the Index register.

(vii) Based Addressing

In this case, the operand's offset is the sum of the contents of the base register + 8-bit/16-bit displacement given in the instruction.

(viii) Based-Index Addressing

Here the contents of the Base register and the contents of the index register are added together to form the effective address. The base register contains a base address and the index register contains an index.

(ix) Relative Addressing

- Here a signed displacement is added to the current value of the Program Code (PC) to form the effective address.
- This mode of addressing is commonly used in branch (or jump) instructions.
- This is also known as PC relative addressing.
- The effective address specified memory location in relation to the current value of the PC.

(x) Relative Indexed Addressing

In this mode of addressing, the contents of the PC and the contents of the index register are added together to form the effective address.

(xi) Page Addressing

- In Paged mode of addressing, the memory is divided into a number of equal length pages.
- The page size is 256 bytes for 8 bit µP and 4 KB for 16-bit µP.
- The µP contains a page register to hold the page number.
- The instruction contains an offset. This offset indicates the address within the page w.r.t. the starting address of the page.
- The advantage of this mode of addressing is that a fewer bits in the instruction are required to indicate the memory address. The result is shorter instruction and faster execution.

(xii) Stack Addressing

In this case, the address of the operand is specified by the Stack Pointer (SP). The contents of the SP are automatically incremented/ decremented after a PUSH/POP.

Interrupts and Exceptions

- When data are ready, an I/O device can interrupt CPU. After completing the current instruction at hand, the CPU attends the I/O device.
- The CPU enters into a subroutine known as Interrupts Service Sub-routine (ISS) to transfer data from the device.
- Each CPU has interrupt lines through which I/O devices can be connected to the CPU.
- When data transfer is over, the CPU returns to the program it was executing.
- An interrupt caused by an external signal applied to an interrupt input line of a CPU is known as hardware interrupt.
- The normal program execution of a µP can also be interrupted by a special instruction in the program. This is known as software interrupt.
- The internal events, which cause the processor to go out of its normal processing sequence, are called Exception.
- The external events caused by external I/O devices, which prevent the further processing are called Interrupts. It handles external asynchronous events.
- Exceptions handle internal abnormal or unusual conditions, which prevent further processing. The processor treats S/W interrupts as exception.

Different type of interrupt in 8085

- In the 8085, as with any CPU that has interrupt capability, there is a method by which the interrupt gets serviced in a timely manner. When the interrupt occurs, and the current instruction that is being processed is finished, the address of the next instruction to be executed is pushed onto the Stack.
- Then a jump is made to a dedicated location where the ISR (Interrupt Service Routine) is located.
- Some interrupts have their own vector, or unique location where it's service routine starts. These are hard coded into the 8085 and can't be changed.
  1. **TRAP** - has highest priority and **cannot** be masked or disabled. A rising-edge pulse will cause a jump to location 0024H.
  2. **RST 7.5**- 2nd priority and can be masked or disabled. Rising-edge pulse will cause a jump to location 7.5 * 8 = 003CH.

     This interrupt is latched internally and must be reset before it can be used again.
  3. **RST 6.5** – 3rd priority and can be masked or disabled. A high logic level will cause a jump to location 6.5 * 8 = 0034H.
  4. **RST 5.5** – 4th priority and can be masked or disabled. A high logic level will cause a jump to location 5.5 * 8 = 002CH.
  5. **INTR** – 5th priority and can be masked or disabled. A high logic level will cause a jump to specific location as follows:

Instruction Cycle

- A program is a sequence of well-defined instructions. Both inputs (i.e. data and the operation to be performed on the data), to the program are stored in the memory.
- The main job of the computer system is to execute these instructions.
- The instruction cycle is the sequence of events that takes place as the instruction is read from the memory and executed.
- Although the **Instruction - Cycle** is called the "**Fetch-Execute cycle**", but a simple instruction cycle can be considered to be consisting of the following 4 steps:

(i)	**Fetch Cycle**	– Fetching the instruction from the memory.
(ii)	**Decode Cycle**	– Decoding the instruction
(iii)	**Execute Cycle**	– Executing the instruction
(iv)	**Store Cycle**	– Storing the result back to the memory

1. **The Fetch Cycle:**

During this cycle, the instruction which is to be executed next, is fetched from the memory to the processor. The steps performed during the fetch cycle are as follows:

i. The Program Counter (PC) keeps track of the memory location of the next instruction.
ii. This address is transferred from PC to MAR.
iii. The instruction is read from the memory.
iv. The instruction thus obtained is stored in the MBR → IR and the PC gets incremented by 1.
v. In the IR, the unique bit patterns that make up the machine language are extracted and sent to the decoder.

2. **The Decode Cycle**

The decode cycle is responsible for recognizing the operation that the bit pattern represents and activating the correct circuitry to perform the operation.

The steps are:

i. The OPCODE of the instruction is first read and then broken up into several micro-instruction, which may contain activities such as reading data from the memory, IO devices or registers.
ii. Hence the read operation may have to performed repeatedly and thus the operand is fetched, and is made available for the execute cycle.

3. **The Execute Cycle**

Once the instruction has been decoded, the operation specified by the OPCODE is performed on user-provided data in ALU.

The following steps are involved:

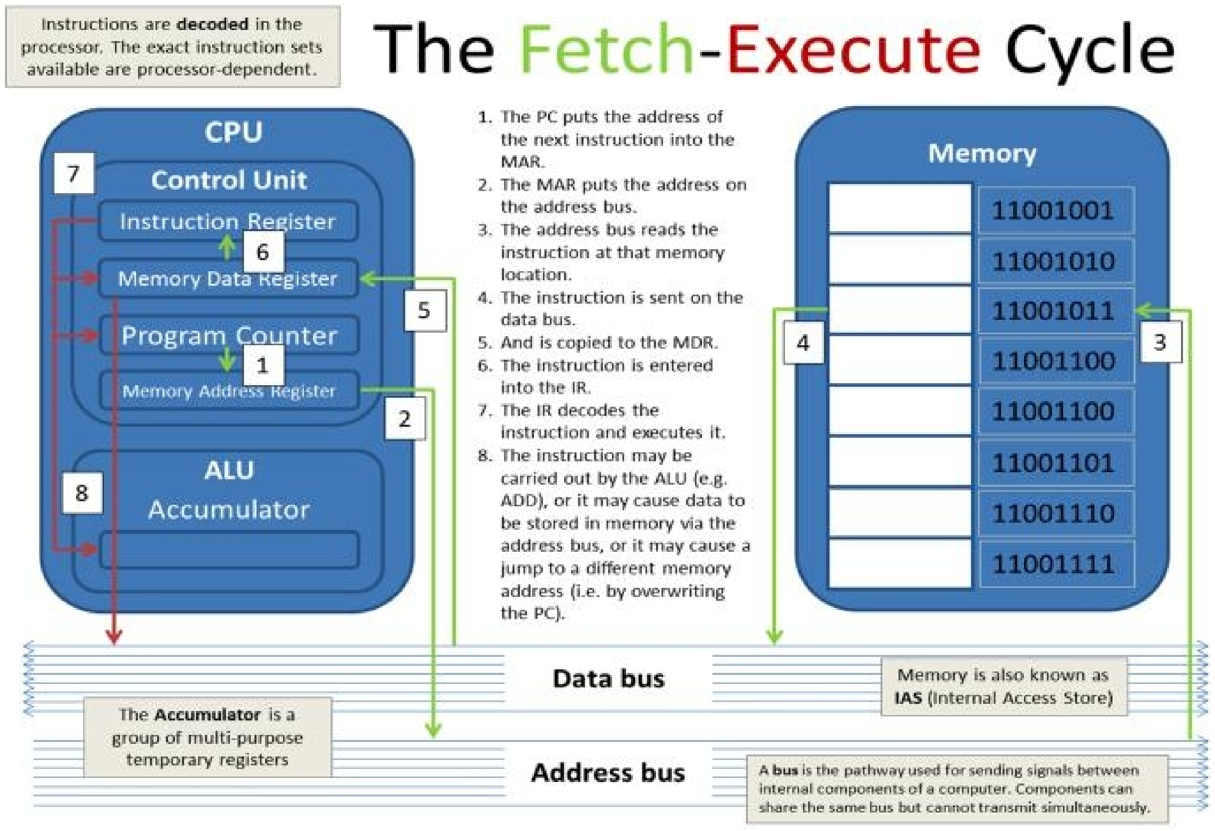
i. The data is fetched into ALU from the memory location pointed by MBR, or available in the register, or in the instruction itself.
ii. The operation specified by the decoded op-code is performed on the data in ALU or it may cause a jump to a different memory location, by overwriting the PC.

4. **Store Cycle**

After the fetch, decode and execute cycles been executed, the results are ready to be stored. The steps involved are:

i. The results from the execution cycle are stored in the MBR.
ii. Then the results from MBR are stored back to the memory, or the IO device.

The entire activity of the Fetch-Execute cycle may be summarized as follows:

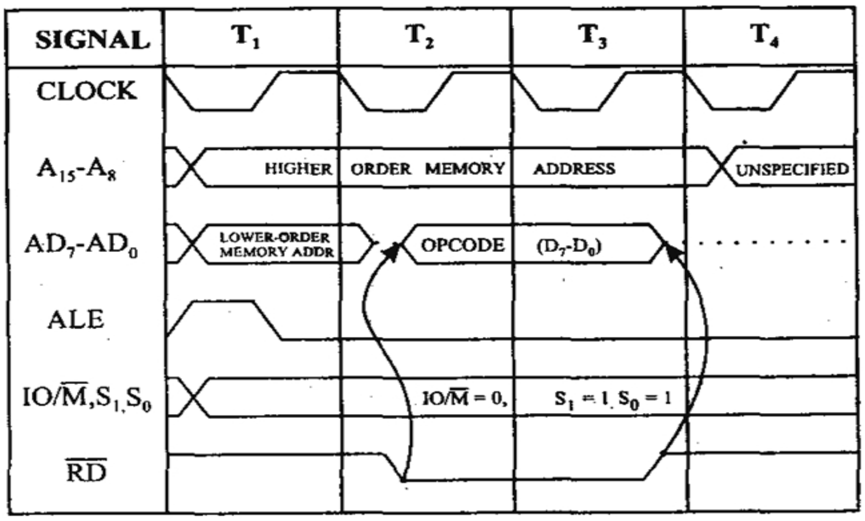

Instructions are **decoded** in the processor. The exact instruction sets available are processor-dependent.
The Fetch-Execute Cycle
CPU
Control Unit
Instruction Register
Memory Data Register
Program Counter
Memory Address Register
ALU
Accumulator
1. The PC puts the address of the next instruction into the MAR.
2. The MAR puts the address on the address bus.
3. The address bus reads the instruction at that memory location.
4. The instruction is sent on the data bus.
5. And is copied to the MDR.
6. The instruction is entered into the IR.
7. The IR decodes the instruction and executes it.
8. The instruction may be carried out by the ALU (e.g. ADD), or it may cause data to be stored in memory via the address bus, or it may cause a jump to a different memory address (i.e. by overwriting the PC).
Memory
11001001
11001010
11001011
11001100
11001100
11001101
11001110
11001111
Data bus
Address bus
Memory is also known as **IAS** (Internal Access Store)
The **Accumulator** is a group of multi-purpose temporary registers
A **bus** is the pathway used for sending signals between internal components of a computer. Components can share the same bus but cannot transmit simultaneously.

Timing diagrams for Fetch-Execution cycles

- **Instruction cycle** is defined, as the time required for completing the execution of an instruction. The 8085-instruction cycle consists of one to six machine cycle or one to six operations.
- **Machine cycle** is defined, as the time required for completing one operation of accessing memory, I/O. or acknowledging an external request. This cycle may consist of three to six T-states.
- **T-state** is defined as one subdivision of the operation performed in one clock period. These subdivisions are internal states synchronized with the system clock, and each T-state is precisely equal to one clock period. The terms **T-state** and **clock period** are often used synonymously.
- The Timing diagram for different operations during Fetch-Execute cycle are given below:

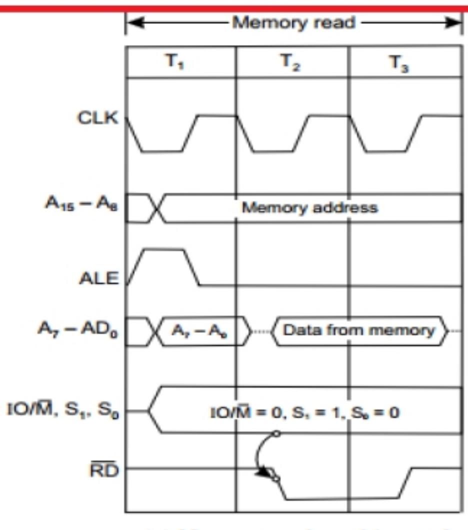
Timing Diagram For opcode fetch Operation

- Here three T-states (T1 - T3)are used for the Fetch operation, and the fourth (T4) for Decode
- It may be noted that, if further read operation is not required, a machine cycle comprises of four T-states only.
- Since the opcode fetch is to be made from memory, IO/$\overline{\text{M}}$ goes low, to indicate the memory operation (where Memory indicates high).
- The ALE (Address Latch Enable), is high in T1, so as to multiplex the Data lines with Address lines to carry the 16-bit memory address from PC.
- Then ALE goes low, so that the instruction (8-bit data) is loaded to the Data bus to be carried from memory to the CPU.
- The $\overline{\text{RD}}$ is high in T1, and goes low in T2, so as to enable the read signal to the memory
- The status bits S_1 =1 and S_0=1, to indicate FETCH operation

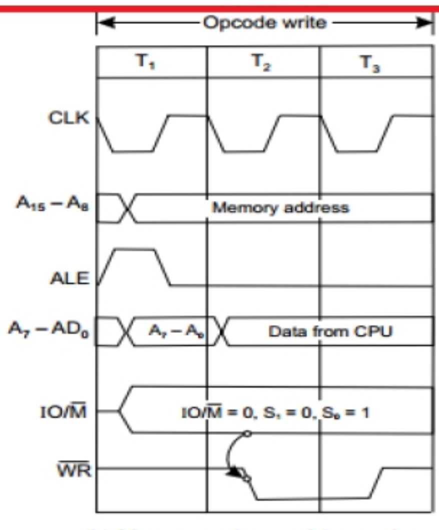


Timing Diagram For Memory Read and write Operation

- Here three T-states (T1 - T3) are used for the read operation.
- Since the read is to be made from memory, IO/$\overline{\text{M}}$ goes low, to indicate the memory operation (where Memory indicates high). In case of IO read, the above bit becomes high.
- The ALE (Address Latch Enable), is high in T1, so as to multiplex the Data lines with Address lines to carry the 16-bit memory address.
- Then ALE goes low, so that the data (8-bit data) is loaded to the Data bus to be carried from memory to the processor during read operation and vice versa.
- The $\overline{\text{RD}}$ is high in T1, and goes low in T2 and T3, so as to enable the read signal to the memory and similarly $\overline{\text{WR}}$ is high in T1 and goes low in T2 and T3.
- The status bits S_1 =1 and S_0=0, to indicate READ operation and S_1 =0 and S_0=1 to indicate WRITE operation.



(a) Memory read machine cycle



(b) Memory write machine cycle

Memory read and write machine cycle