Unit-4

Instruction Timing and Cycle

An instruction cycle (also known as the fetch–decode–execute cycle or the fetch-execute cycle) is the basic operational process of a computer. It is the process by which a computer retrieves a program instruction from its memory, determines what actions the instruction dictates, and carries out those actions. This cycle is repeated continuously by a computer's central processing unit (CPU), from boot-up to when the computer is shut down.

In simpler CPU's the instruction cycle is executed sequentially, each instruction being processed before the next one is started. In most modern CPU's the instruction cycles are instead executed concurrently, and often in parallel, through an instruction pipeline: the next instruction starts being processed before the previous instruction has finished, which is possible because the cycle is broken up into separate steps.

1. **Initiating the cycle**
   - The cycle starts immediately when power is applied to the system using an initial PC value that is predefined for the system architecture
   - Typically this address points to instructions in a read-only memory (ROM) (not the random access memory or RAM) which begins the process of loading the operating system.

2. **Fetch the Instruction**

   Step 1 of the Instruction Cycle is called the Fetch Cycle. This step is the same for each instruction.

   i. The CPU sends PC to the MAR and sends a READ command on the control bus
   ii. In response to the read command (with address equal to PC), the memory returns the data stored at the memory location indicated by PC on the data bus.
   iii. The CPU copies the data from the data bus into its MDR (also known as MBR)
   iv. A fraction of a second later, the CPU copies the data from the MDR to the Instruction Register (IR)
   v. The PC is incremented so that it points to the following instruction in memory. This step prepares the CPU for the next cycle. The Control Unit fetches the instruction's address from the Memory Unit

3. **Decode the Instruction**
   - Step 2 of the instruction Cycle is called the Decode Cycle. The decoding process allows the CPU to determine what instruction is to be performed, so that the CPU can tell how many operands it needs to fetch in order to perform the instruction.
   - The opcode fetched from the memory is decoded for the next steps and moved to the appropriate registers. The decoding is done by the CPU's Control Unit.

4. **Read the effective address**
   - Step 3 is deciding which operation it is. If this is a Memory operation - in this step the computer checks if it's a direct or indirect memory operation:
   - Direct memory instruction - Nothing is being done.
   - Indirect memory instruction - The effective address is being read from the memory. If this is a I/O or Register instruction - the computer checks its kind and executes the instruction.

5. **Execute the Instruction**
    - Step 4 of the Instruction Cycle is the Execute Cycle. Here, the function of the instruction is performed.
    - If the instruction involves arithmetic or logic, the Arithmetic Logic Unit is utilized. This is the only stage of the instruction cycle that is useful from the perspective of the end user.
    - Everything else is overhead required to make the execute stage happen.

Machine cycle

A *machine cycle*, also called a *processor cycle* or a *instruction cycle*, is the basic operation performed by a central processing unit (CPU). A CPU is the main logic unit of a computer.

A machine cycle consists of a sequence of three steps that is performed continuously and at a rate of millions per second while a computer is in operation. They are *fetch*, *decode* and *execute*. There also is a fourth step, *store*, in which input and output from the other three phases is stored in memory for later use; however, no actual processing is performed during this step.

In the fetch step, the *control unit* requests that main memory provide it with the instruction that is stored at the *address* (i.e., location in memory) indicated by the control unit's *program counter*.

The control unit is a part of the CPU that also decodes the instruction in the *instruction register*. A *register* is a very small amount of very fast memory that is built into the CPU in order to speed up its operations by providing quick access to commonly used values; instruction registers are registers that hold the instruction being executed by the CPU. Decoding the instructions in the instruction register involves breaking the *operand field* into its components based on the instructions *opcode*.

Opcode (an abbreviation of *operation code*) is the portion of a *machine language* instruction that specifies what operation is to be performed by the CPU. Machine language, also called *machine code*, refers to instructions coded in patterns of bits (i.e., zeros and ones) that are directly readable and executable by a CPU.

A program counter, also called the *instruction pointer* in some computers, is a register that indicates where the computer is in its instruction sequence. It holds either the address of the instruction currently being executed or the address of the next instruction to be executed, depending on the details of the particular computer. The program counter is automatically incremented for each machine cycle so that instructions are normally retrieved sequentially from memory.

The control unit places these instructions into its instruction register and then increments the program counter so that it contains the address of the next instruction stored in memory. It then executes the instruction by activating the appropriate circuitry to perform the requested task. As soon as the instruction has been executed, it restarts the machine cycle, beginning with the fetch step.

T states

One complete cycle of clock is called as T-state as shown in the above figure. A T-state is measured from the falling edge of one clock pulse to the falling edge of the next clock pulse. Various versions of 8086 have maximum clock frequency from 5MHz to 10MHz. Hence the minimum time for one T-state is between 100 to 200 nsec.

Instruction Execution And Timing Diagram:

Each instruction in 8085 microprocessor consists of two part- operation code (opcode) and operand. The opcode is a command such as ADD and the operand is an object to be operated on, such as a byte or the content of a register.

Instruction Cycle: The time taken by the processor to complete the execution of an instruction. An instruction cycle consists of one to six machine cycles.

Machine Cycle: The time required to complete one operation; accessing either the memory or I/O device. A machine cycle consists of three to six T-states.

T-State: Time corresponding to one clock period. It is the basic unit to calculate execution of instructions or programs in a processor.

To execute a program, 8085 performs various operations as:

- Opcode fetch
- Operand fetch
- Memory read/write
- I/O read/write

External communication functions are:

- Memory read/write
- I/O read/write
- Interrupt request acknowledge
-

Opcode Fetch Machine Cycle:

It is the first step in the execution of any instruction. The timing diagram of this cycle is given below

The following points explain the various operations that take place and the signals that are changed during the execution of opcode fetch machine cycle:

T1 clock cycle

i. The content of PC is placed in the address bus; AD0 - AD7 lines contains lower bit address and A8 – A15 contains higher bit address.
ii. M IO/ signal is low indicating that a memory location is being accessed. S1 and S0 also changed to the levels as indicated in Table 1.
iii. ALE is high, indicates that multiplexed AD0 – AD7 act as lower order bus.

T2 clock cycle

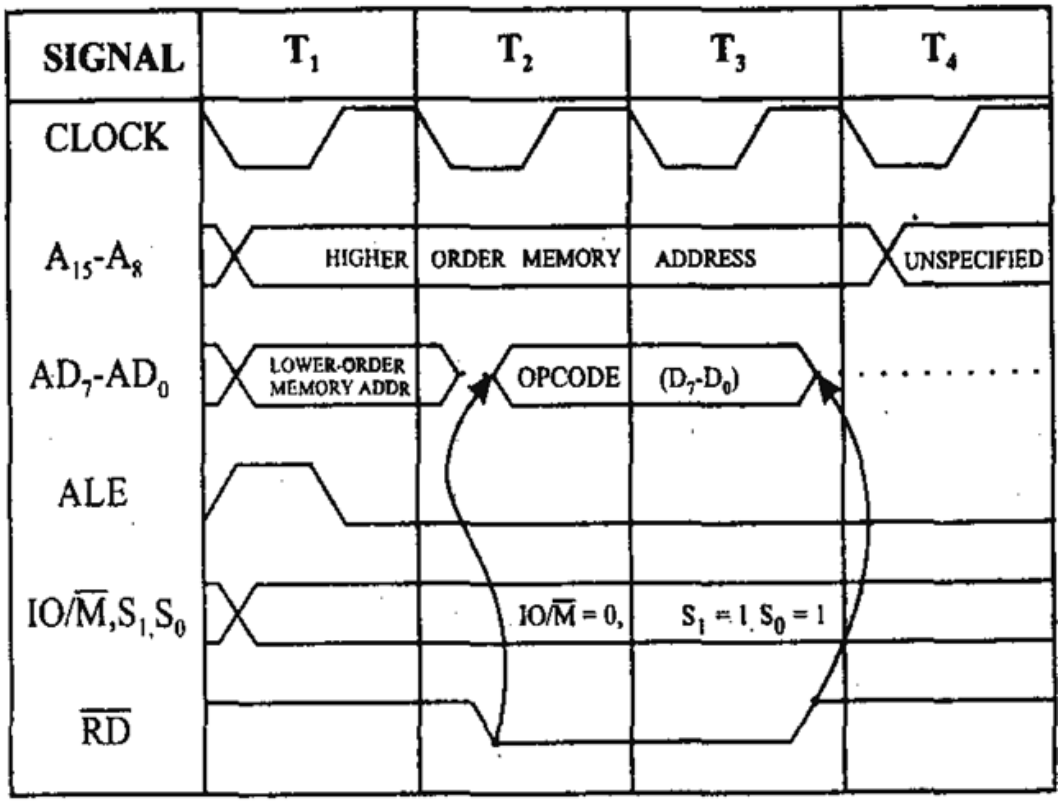
i. Multiplexed address bus is now changed to data bus.
ii. The RD signal is made low by the processor. This signal makes the memory device load the data bus with the contents of the location addressed by the processor.

T3 clock cycle

i. The opcode available on the data bus is read by the processor and moved to the instruction register. ii. The RD signal is deactivated by making it logic 1.

T4 clock cycle

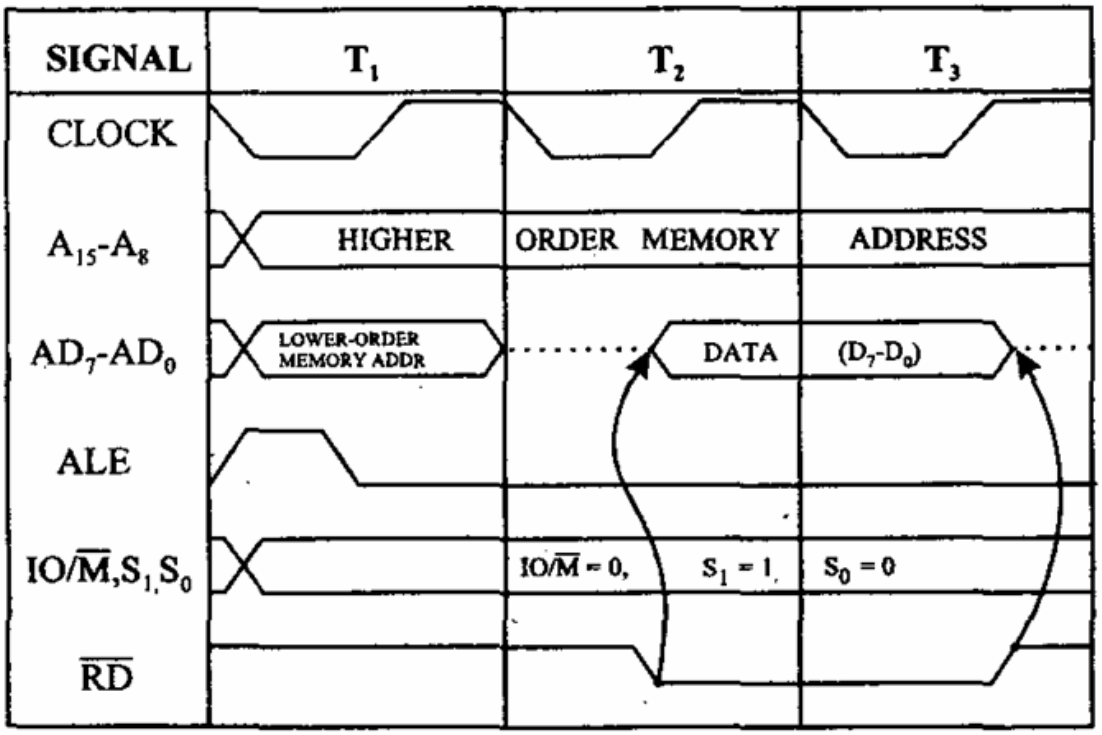
i. The processor decode the instruction in the instruction register and generate the necessary control signals to execute the instruction. Based on the instruction further operations such as fetching, writing into memory etc takes place.



Timing diagram for opcode fetch cycle

Memory Read Machine Cycle:

The memory read cycle is executed by the processor to read a data byte from memory. The machine cycle is exactly same to opcode fetch except: a) It has three T-states b) The S0 signal is set to 0. The timing diagram of this cycle is given in Fig.



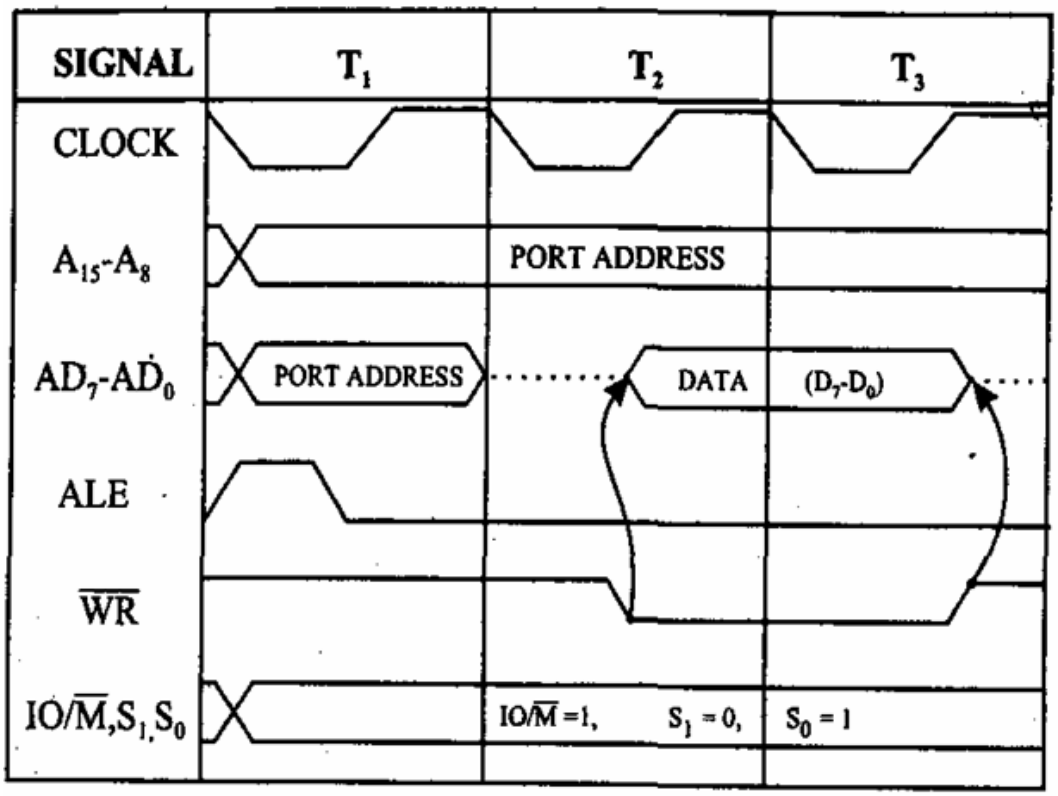
Timing diagram for memory read machine cycle

Memory Write Machine Cycle:

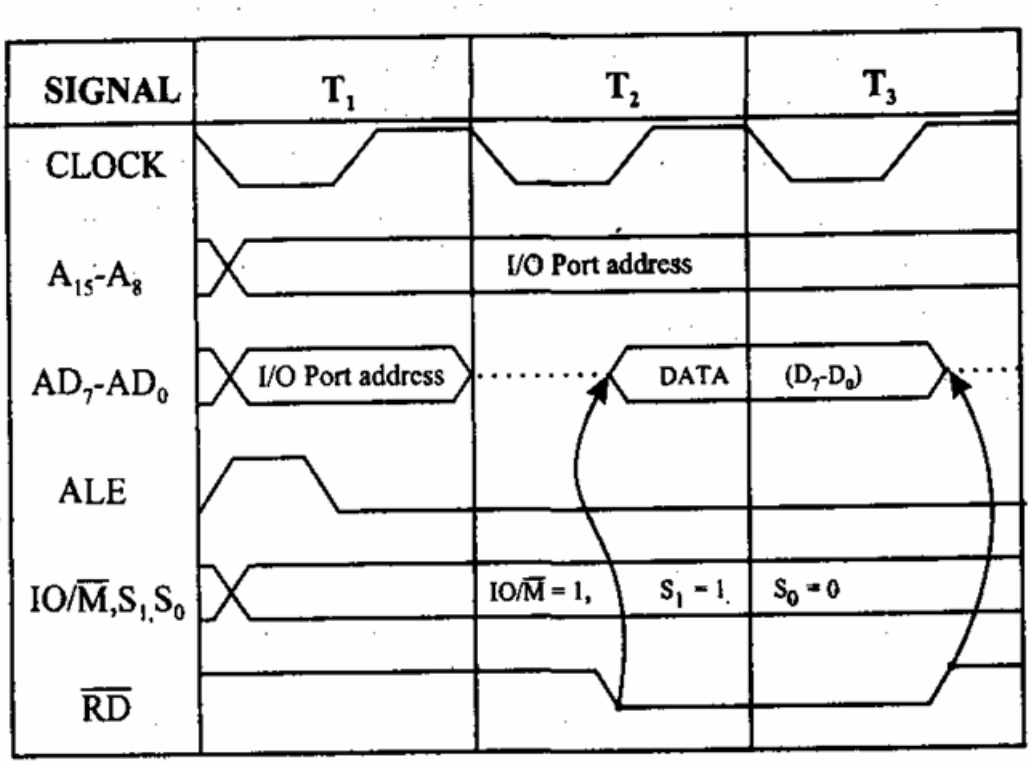
The memory write cycle is executed by the processor to write a data byte in a memory location. The processor takes three T-states and WR signal is made low. The timing diagram of this cycle is given below.

I/O Read Cycle:

The I/O read cycle is executed by the processor to read a data byte from I/O port or from peripheral, which is I/O mapped in the system. The 8-bit port address is placed both in the lower and higher order address bus. The processor takes three T-states to execute this machine cycle. The timing diagram of this cycle is given below



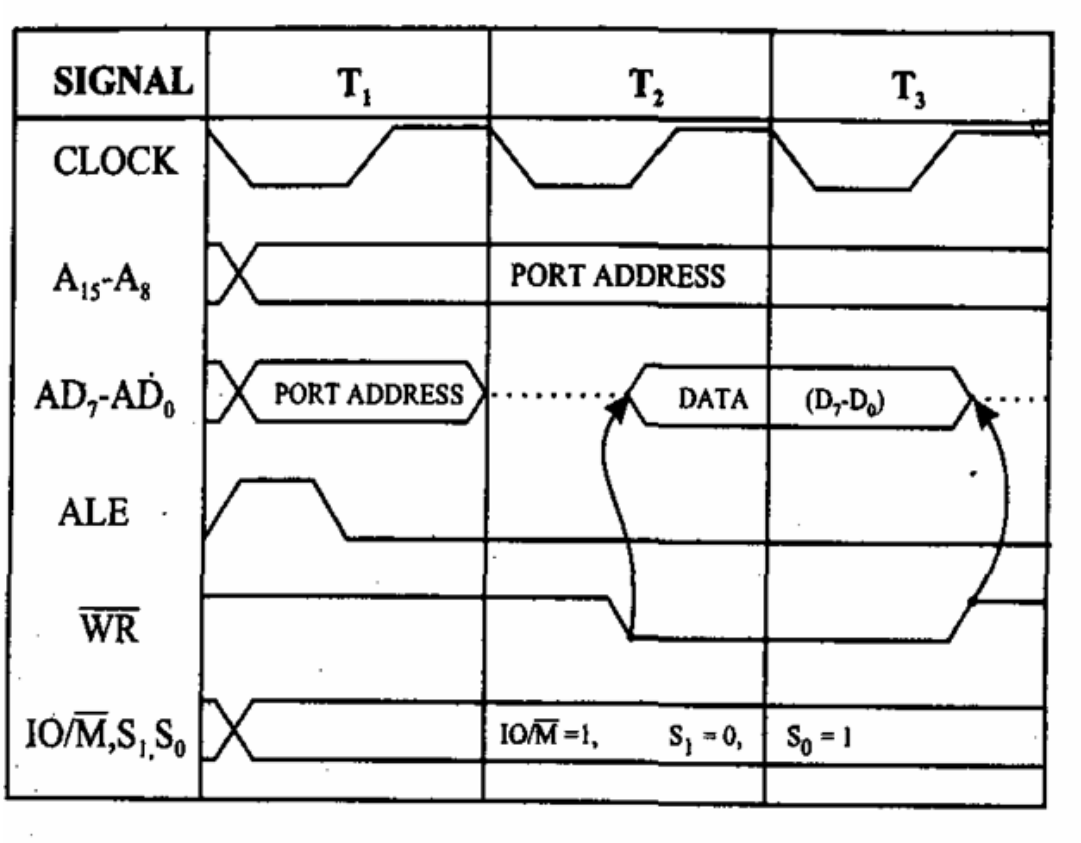
Timing diagram for memory write machine cycle



Timing diagram I/O read machine cycle

I/O Write Cycle:

The I/O write cycle is executed by the processor to write a data byte to I/O port or to a peripheral, which is I/O mapped in the system. The processor takes three T-states to execute this machine cycle. The timing diagram of this cycle is given in dia.



Timing diagram I/O write machine cycle

Ex: Timing diagram for IN 80H.

Assembly languages

An assembly (or assembler) language, often abbreviated asm, is a low-level programming language for a computer, or other programmable device, in which there is a very strong (but often not one-to-one) correspondence between the language and the architecture's machine code instructions. Each assembly language is specific to a particular computer architecture. In contrast, most high-level programming languages are generally portable across multiple architectures but require interpreting or compiling. Assembly language may also be called symbolic machine code.

Machine language

Machine code or machine language is a set of instructions executed directly by a computer's central processing unit (CPU). Each instruction performs a very specific task, such as a load, a jump, or an ALU operation on a unit of data in a CPU register or memory. Every program directly executed by a CPU is made up of a series of such instructions. (The phrase 'directly executed' needs some clarification; machine code is by definition the lowest level of programming detail visible to the programmer, but internally many processors use microcode or optimize and transform machine code instructions into sequences of micro-ops in a sophisticated way.)

Mnemonics

Mnemonics allow users to access quickly a wide variety of commands, services, programs and functions without the need to type out extended phrases. One example of a mnemonic code is the term "inc," which on an Intel microprocessor refers to the command "increase by one." Rather than type the entire phrase, the letters "inc" can be entered. Mnemonic code derives from the concept of traditional mnemonics in which abbreviations, rhymes or simple stories are used to help people remember information.

Instruction Set and Execution In 8085

Based on the design of the ALU and decoding unit, the microprocessor manufacturer provides instruction set for every microprocessor. The instruction set consists of both machine code and mnemonics.

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called instruction set. Microprocessor instructions can be classified based on the parameters such functionality, length and operand addressing.

Classification based on functionality:

I. **Data transfer operations:** This group of instructions copies data from source to destination. The content of the source is not altered.
II. **Arithmetic operations:** Instructions of this group perform operations like addition, subtraction, increment & decrement. One of the data used in arithmetic operation is stored in accumulator and the result is also stored in accumulator.
III. **Logical operations:** Logical operations include AND, OR, EXOR, NOT. The operations like AND, OR and EXOR uses two operands, one is stored in accumulator and other can be any register or

memory location. The result is stored in accumulator. NOT operation requires single operand, which is stored in accumulator.

IV. **Branching operations:** Instructions in this group can be used to transfer program sequence from one memory location to another either conditionally or unconditionally. V. Machine control operations: Instruction in this group control execution of other instructions and control operations like interrupt, halt etc.

Classification based on length:

I. **One-byte instructions:** Instruction having one byte in machine code. Examples are depicted in Table 2

II. **Two-byte instructions:** Instruction having two byte in machine code. Examples are depicted in Table 3.

III. **Three-byte instructions:** Instruction having three byte in machine code. Examples are depicted in Table 4.

Table 2 Examples of one byte instructions

Opcode	Operand	Machine code/Hex code
MOV	A, B	78
ADD	M	86

Table 3 Examples of two byte instructions

Opcode	Operand	Machine code/Hex code	Byte description
MVI A,	7FH	3E	First byte
		7F	Second byte
ADI	0FH	C6	First byte
		0F	Second byte

Table 4 Examples of three byte instructions

Opcode	Operand	Machine code/Hex code	Byte description
JMP	9050H	C3	First byte
		50	Second byte
		90	Third byte
LDA	8850H	3A	First byte
		50	Second byte
		88	Third byte

Addressing Modes in Instructions:

The process of specifying the data to be operated on by the instruction is called addressing. The various formats for specifying operands are called addressing modes.

The 8085 has the following five types of addressing:

I. Immediate addressing

II. Memory direct addressing
III. Register direct addressing
IV. Indirect addressing
V. Implicit addressing

Immediate Addressing:
In this mode, the operand given in the instruction - a byte or word – transfers to the destination register or memory location.
Ex: MVI A, 9AH

- The operand is a part of the instruction.
- The operand is stored in the register mentioned in the instruction.

Memory Direct Addressing:
Memory direct addressing moves a byte or word between a memory location and register. The memory location address is given in the instruction.
Ex: LDA 850FH
This instruction is used to load the content of memory address 850FH in the accumulator.

Register Direct Addressing:
Register direct addressing transfer a copy of a byte or word from source register to destination register.
Ex: MOV B, C
It copies the content of register C to register B.

Indirect Addressing:
Indirect addressing transfers a byte or word between a register and a memory location.
Ex: MOV A, M
Here the data is in the memory location pointed to by the contents of HL pair. The data is moved to the accumulator.

Implicit Addressing
In this addressing mode the data itself specifies the data to be operated upon.
Ex: CMA
The instruction complements the content of the accumulator. No specific data or operand is mentioned in the instruction.

INSTRUCTION SET OF 8085

Data Transfer Instructions

Opcode	Operand	Description
Copy from source to destination		
MOV	Rd, Rs M, Rs Rd, M	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. Example: MOV B, C or MOV B, M
Move immediate 8-bit		
MVI	Rd, data M, data	The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: MVI B, 57H or MVI M, 57H
Load accumulator		
LDA	16-bit address	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034H
Load accumulator indirect		
LDAX	B/D Reg. pair	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. Example: LDAX B
Load register pair immediate		
LXI	Reg. pair, 16-bit data	The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034H or LXI H, XYZ
Load H and L registers direct		
LHLD	16-bit address	The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. Example: LHLD 2040H

Store accumulator direct

STA	16-bit address	The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example: STA 4350H

Store accumulator indirect

STAX	Reg. pair	The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered. Example: STAX B

Store H and L registers direct

SHLD	16-bit address	The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example: SHLD 2470H

Exchange H and L with D and E

XCHG	None	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. Example: XCHG

Copy H and L registers to the stack pointer

SPHL	None	The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered. Example: SPHL

Exchange H and L with top of stack

XTHL	None	The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered. Example: XTHL

Push register pair onto stack

PUSH	Reg. pair	The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the high-order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location. Example: PUSH B or PUSH A

Pop off stack to register pair

POP	Reg. pair	The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1. Example: POP H or POP A

Output data from accumulator to a port with 8-bit address

OUT	8-bit port address	The contents of the accumulator are copied into the I/O port specified by the operand. Example: OUT F8H

Input data to accumulator from a port with 8-bit address

IN	8-bit port address	The contents of the input port designated in the operand are read and loaded into the accumulator. Example: IN 8CH

Arithmetic Instructions

Opcode	Operand	Description
Add register or memory to accumulator		
ADD	R M	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADD B or ADD M
Add register to accumulator with carry		
ADC	R M	The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADC B or ADC M
Add immediate to accumulator		
ADI	8-bit data	The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ADI 45H
Add immediate to accumulator with carry		
ACI	8-bit data	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ACI 45H
Add register pair to H and L registers		
DAD	Reg. pair	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. Example: DAD H

Subtract register or memory from accumulator

SUB	R M	The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. Example: SUB B or SUB M

Subtract source and borrow from accumulator

SBB	R M	The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. Example: SBB B or SBB M

Subtract immediate from accumulator

SUI	8-bit data	The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction. Example: SUI 45H

Subtract immediate from accumulator with borrow

SBI	8-bit data	The 8-bit data (operand) and the Borrow flag are subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtracion. Example: SBI 45H

Increment register or memory by 1

INR	R M	The contents of the designated register or memory) are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: INR B or INR M

Increment register pair by 1

INX	R	The contents of the designated register pair are incremented by 1 and the result is stored in the same place. Example: INX H

Decrement register or memory by 1

DCR R
M

The contents of the designated register or memory are decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.
Example: DCR B or DCR M

Decrement register pair by 1

DCX R

The contents of the designated register pair are decremented by 1 and the result is stored in the same place.
Example: DCX H

Decimal adjust accumulator

DAA none

The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation.

If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits.

If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits.

Example: DAA

Branching Instructions

Opcode	Operand	Description
Jump unconditionally		
JMP	16-bit address	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Example: JMP 2034H or JMP XYZ

Jump conditionally

Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below.
Example: JZ 2034H or JZ XYZ

Opcode	Description	Flag Status
JC	Jump on Carry	CY = 1
JNC	Jump on no Carry	CY = 0
JP	Jump on positive	S = 0
JM	Jump on minus	S = 1
JZ	Jump on zero	Z = 1
JNZ	Jump on no zero	Z = 0
JPE	Jump on parity even	P = 1
JPO	Jump on parity odd	P = 0

Unconditional subroutine call

CALL 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack. Example: CALL 2034H or CALL XYZ

Call conditionally

Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack. Example: CZ 2034H or CZ XYZ

Opcode	Description	Flag Status
CC	Call on Carry	CY = 1
CNC	Call on no Carry	CY = 0
CP	Call on positive	S = 0
CM	Call on minus	S = 1
CZ	Call on zero	Z = 1
CNZ	Call on no zero	Z = 0
CPE	Call on parity even	P = 1
CPO	Call on parity odd	P = 0

Return from subroutine unconditionally

RET	none	The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address. Example: RET

Return from subroutine conditionally

Operand: none

The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.
Example: RZ

Opcode	Description	Flag Status
RC	Return on Carry	CY = 1
RNC	Return on no Carry	CY = 0
RP	Return on positive	S = 0
RM	Return on minus	S = 1
RZ	Return on zero	Z = 1
RNZ	Return on no zero	Z = 0
RPE	Return on parity even	P = 1
RPO	Return on parity odd	P = 0

Load program counter with HL contents

PCHL none — The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte.
Example: PCHL

Restart

RST 0-7 — The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are:

Instruction	Restart Address
RST 0	0000H
RST 1	0008H
RST 2	0010H
RST 3	0018H
RST 4	0020H
RST 5	0028H
RST 6	0030H
RST 7	0038H

The 8085 has four additional interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are:

Interrupt	Restart Address
TRAP	0024H
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

Logical Instructions

Opcode	Operand	Description
Compare register or memory with accumulator		

CMP	R
	M
		The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved . The result of the comparison is shown by setting the flags of the PSW as follows:
		if (A) < (reg/mem): carry flag is set if
		(A) = (reg/mem): zero flag is set
		if (A) > (reg/mem): carry and zero flags are reset
		Example: CMP B or CMP M

Compare immediate with accumulator		

CPI	8-bit data	The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:
		if (A) < data: carry flag is set if
		(A) = data: zero flag is set
		if (A) > data: carry and zero flags are reset Example:
		CPI 89H

Logical AND register or memory with accumulator		
ANA	R M	The contents of the accumulator are logically ANDed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. Example: ANA B or ANA M
Logical AND immediate with accumulator		
ANI	8-bit data	The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. Example: ANI 86H

Exclusive OR register or memory with accumulator

XRA	R M	The contents of the accumulator are Exclusive ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: XRA B or XRA M

Exclusive OR immediate with accumulator

XRI	8-bit data	The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: XRI 86H

Logical OR register or memory with accumulaotr

ORA	R M	The contents of the accumulator are logically ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: ORA B or ORA M

Logical OR immediate with accumulator

ORI	8-bit data	The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: ORI 86H

Rotate accumulator left

RLC	none	Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7. S, Z, P, AC are not affected. Example: RLC

Rotate accumulator right

RRC	none	Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0. S, Z, P, AC are not affected. Example: RRC

Rotate accumulator left through carry

RAL	none	Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7. S, Z, P, AC are not affected. Example: RAL

Rotate accumulator right through carry

RAR	none	Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0. S, Z, P, AC are not affected. Example: RAR

Complement accumulator

CMA	none	The contents of the accumulator are complemented. No flags are affected. Example: CMA

Complement carry

CMC	none	The Carry flag is complemented. No other flags are affected. Example: CMC

Set Carry

STC	none	The Carry flag is set to 1. No other flags are affected. Example: STC

Control Instructions

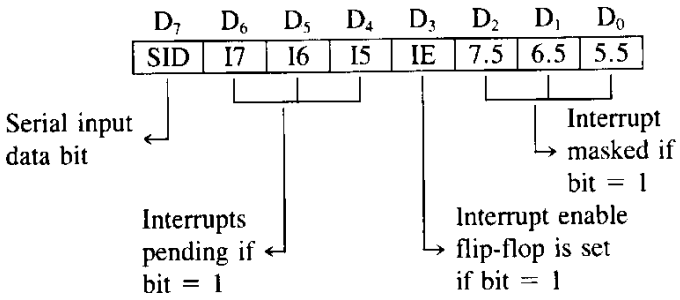
Opcode	Operand	Description
No operation		
NOP	none	No operation is performed. The instruction is fetched and decoded. However no operation is executed. Example: NOP
Halt and enter wait state		
HLT	none	The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. Example: HLT
Disable interrupts		
DI	none	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. Example: DI
Enable interrupts		
EI	none	The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts (except TRAP). Example: EI

Read interrupt mask

RIM none

This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations.

Example: RIM



Set interrupt mask

SIM none

This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interprets the accumulator contents as follows.

Example: SIM