LECTURE NOTES ON

# MICROPROCESSOR AND MICROCONTROLLER

## 4TH SEMESTER ETC


Prepared By: Sasmita Das

GOVERNMENT POLYTECHNIC, DHENKANAL

## 1. INTRODUCTION TO MICROPROCESSOR AND MICROCOMPUTER ARCHITECTURE:

A *microprocessor* is a programmable electronics chip that has computing and decision making capabilities similar to central processing unit of a computer. Any microprocessor-based systems having limited number of resources are called *microcomputers*. Nowadays, microprocessor can be seen in almost all types of electronics devices like mobile phones, printers, washing machines etc. Microprocessors are also used in advanced applications like radars, satellites and flights. Due to the rapid advancements in electronic industry and large scale integration of devices results in a significant cost reduction and increase application of microprocessors and their derivatives.

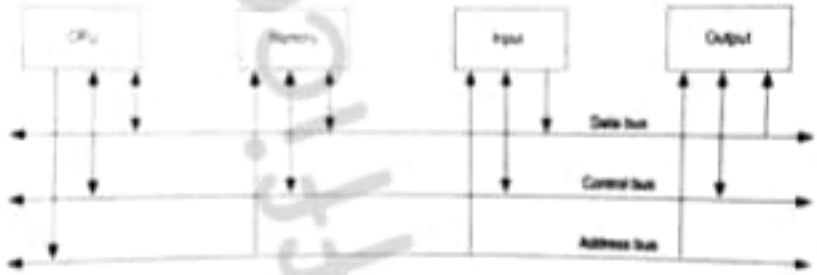



Fig.1 Microprocessor-based system

- **Bit**: A bit is a single binary digit.
- **Word**: A word refers to the basic data size or bit size that can be processed by the arithmetic and logic unit of the processor. A 16-bit binary number is called a word in a 16-bit processor.
- **Bus**: A bus is a group of wires/lines that carry similar information.
- **System Bus**: The system bus is a group of wires/lines used for communication between the microprocessor and peripherals.
- **Memory Word**: The number of bits that can be stored in a register or memory element is called a memory word.
- **Address Bus**: It carries the address, which is a unique binary pattern used to identify a memory location or an I/O port. For example, an eight bit address bus has eight lines and thus it can address $2^8 = 256$ different locations. The locations in hexadecimal format can be written as 00H – FFH.
- **Data Bus**: The data bus is used to transfer data between memory and processor or between I/O device and processor. For example, an 8-bit processor will generally have an 8-bit data bus and a 16-bit processor will have 16-bit data bus.
- **Control Bus**: The control bus carry control signals, which consists of signals for selection of memory or I/O device from the given address, direction of data transfer and synchronization of data transfer in case of slow devices.

A typical microprocessor consists of arithmetic and logic unit (ALU) in association with control unit to process the instruction execution. Almost all the microprocessors are based on the principle of store-program concept. In *store-program concept*, programs or instructions are sequentially stored in the memory locations that are to be executed. To do any task using a microprocessor, it is to be programmed by the user. So the programmer must have idea about its internal resources, features and supported instructions. Each microprocessor has a set of instructions, a list which is provided by the microprocessor manufacturer. The instruction set of a microprocessor is provided in two forms: *binary machine code and mnemonics*.

Microprocessor communicates and operates in binary numbers 0 and 1. The set of instructions in the form of binary patterns is called a *machine language* and it is difficult for us to understand. Therefore, the binary patterns are given abbreviated names, called mnemonics, which forms the *assembly language*. The conversion of assembly-level language into binary machine-level language is done by using an application called *assembler*.

Technology Used:

The semiconductor manufacturing technologies used for chips are:

- Transistor-Transistor Logic (TTL)
- Emitter Coupled Logic (ECL)
- Complementary Metal-Oxide Semiconductor (CMOS)

Classification of Microprocessors:

Based on their specification, application and architecture microprocessors are classified.

*Based on size of data bus:*

- 4-bit microprocessor
- 8-bit microprocessor
- 16-bit microprocessor
- 32-bit microprocessor

*Based on application:*

- General-purpose microprocessor- used in general computer system and can be used by programmer for any application. Examples, 8085 to Intel Pentium.
- Microcontroller- microprocessor with built-in memory and ports and can be programmed for any generic control application. Example, 8051.
- Special-purpose processors- designed to handle special functions required for an application. Examples, digital signal processors and application-specific integrated circuit (ASIC) chips.

*Based on architecture:*

- Reduced Instruction Set Computer (RISC) processors
- Complex Instruction Set Computer (CISC) processors

## 2. 8085 MICROPROCESSOR ARCHITECTURE

The 8085 microprocessor is an 8-bit processor available as a 40-pin IC package and uses +5 V for power. It can run at a maximum frequency of 3 MHz. Its data bus width is 8-bit and address bus width is 16-bit, thus it can address $2^{16}$ = 64 KB of memory. The internal architecture of 8085 is shown is Fig. 2.

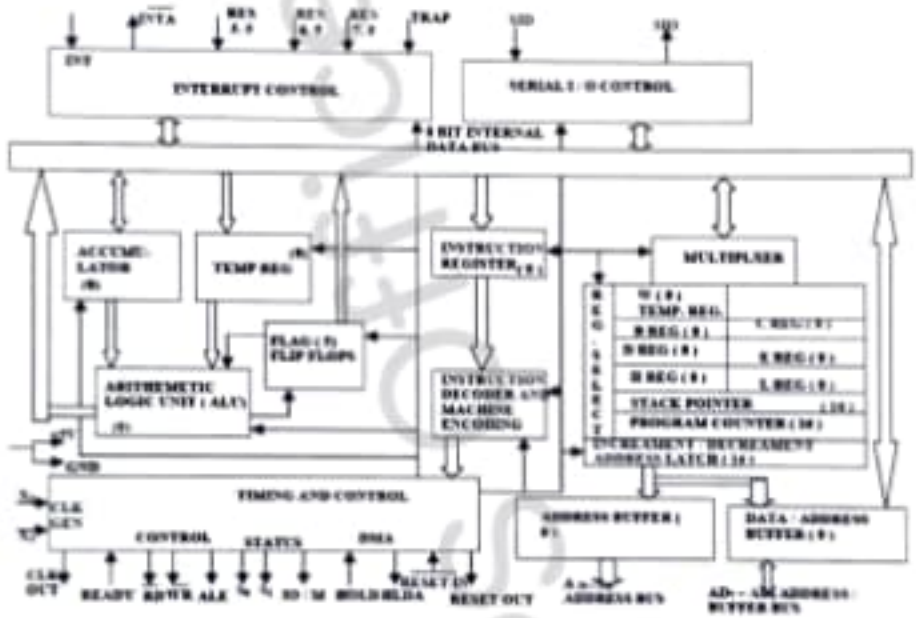



Fig. 2 Internal Architecture of 8085

### Arithmetic and Logic Unit

The ALU performs the actual numerical and logical operations such as Addition (ADD), Subtraction (SUB), AND, OR etc. It uses data from memory and from Accumulator to perform operations. The results of the arithmetic and logical operations are stored in the accumulator.

### Registers

The 8085 includes six registers, one accumulator and one flag register, as shown in Fig. 3. In addition, it has two 16-bit registers: stack pointer and program counter. They are briefly described as follows.

The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H and L. they can be combined as register pairs - BC, DE and HL to perform some

16- bit operations. The programmer can use these registers to store or copy data into the register by using data copy instructions.

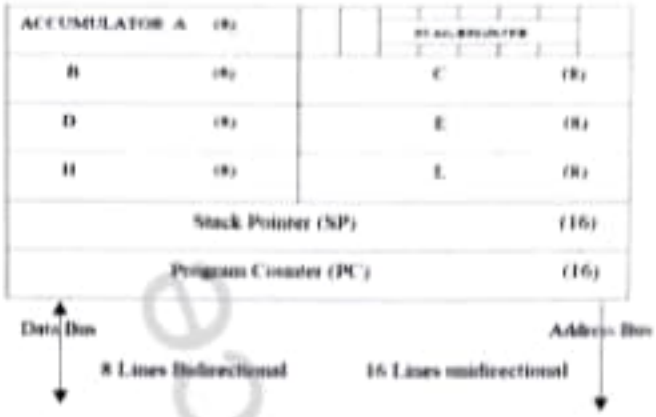



Fig. 3 Register organisation

Accumulator

The accumulator is an 8-bit register that is a part of ALU. This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

Flag register

The ALU includes five flip-flops, which are set or reset after an operation according to data condition of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P) and Auxiliary Carry (AC) flags. Their bit positions in the flag register are shown in Fig. 4. The microprocessor uses these flags to test data conditions.

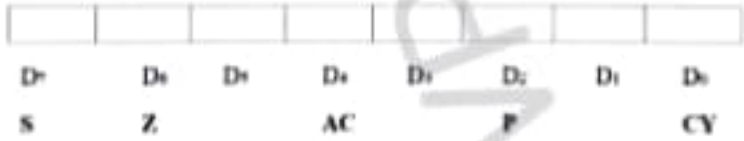



Fig. 4 Flag register

For example, after an addition of two numbers, if the result in the accumulator is larger than 8-bit, the flip-flop uses to indicate a carry by setting CY flag to 1. When an arithmetic operation results in zero, Z flag is set to 1. The S flag is just a copy of the bit D7 of the accumulator. A negative number has a 1 in bit D7 and a positive number has a 0 in 2's complement representation. The AC flag is set to 1, when a carry result from bit D3 and passes to bit D4. The P flag is set to 1, when the result in accumulator contains even number of 1s.

Program Counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte is being fetched, the program counter is automatically incremented by one to point to the next memory location.

Stack Pointer (SP)

The stack pointer is also a 16-bit register, used as a memory pointer. It points to a memory location in R/W memory, called stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

Instruction Register/Decoder

It is an 8-bit register that temporarily stores the current instruction of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and decodes or interprets the instruction. Decoded instruction then passed to next stage.

Control Unit

Generates signals on data bus, address bus and control bus within microprocessor to carry out the instruction, which has been decoded. Typical buses and their timing are described as follows:

- *Data Bus*: Data bus carries data in binary form between microprocessor and other external units such as memory. It is used to transmit data i.e. information, results of arithmetic etc between memory and the microprocessor. Data bus is bidirectional in nature. The data bus width of 8085 microprocessor is 8-bit i.e. $2^8$ combination of binary digits and are typically identified as D0 – D7. Thus size of the data bus determines what arithmetic can be done. If only 8-bit wide then largest number is 11111111 (255 in decimal). Therefore, larger numbers have to be broken down into chunks of 255. This slows microprocessor.
- *Address Bus*: The address bus carries addresses and is one way bus from microprocessor to the memory or other devices. 8085 microprocessor contain 16-bit address bus and are generally identified as A0 - A15. The higher order address lines (A8 – A15) are unidirectional and the lower order lines (A0 – A7) are multiplexed (time-shared) with the eight data bits (D0 – D7) and hence, they are bidirectional.
- *Control Bus*: Control bus are various lines which have specific functions for coordinating and controlling microprocessor operations. The control bus carries control signals partly unidirectional and partly bidirectional. The following control and status signals are used by 8085 processor:
    1. ALE (output): Address Latch Enable is a pulse that is provided when an address appears on the AD0 – AD7 lines, after which it becomes 0.

II. $\overline{RD}$ (active low output): The Read signal indicates that data are being read from the selected I/O or memory device and that they are available on the data bus.

III. $\overline{WR}$ (active low output): The Write signal indicates that data on the data bus are to be written into a selected memory or I/O location.

IV. $IO/\overline{M}$ (output): It is a signal that distinguished between a memory operation and an I/O operation. When $IO/\overline{M} = 0$ it is a memory operation and $IO/\overline{M} = 1$ it is an I/O operation.

V. S1 and S0 (output): These are status signals used to specify the type of operation being performed; they are listed in Table 1.

Table 1 Status signals and associated operations

| S1 | S0 | States |
|---|---|---|
| 0 | 0 | Halt |
| 0 | 1 | Write |
| 1 | 0 | Read |
| 1 | 1 | Fetch |

The schematic representation of the 8085 bus structure is as shown in Fig. 5. The microprocessor performs primarily four operations:

I. Memory Read: Reads data (or instruction) from memory.
II. Memory Write: Writes data (or instruction) into memory.
III. I/O Read: Accepts data from input device.
IV. I/O Write: Sends data to output device.

The 8085 processor performs these functions using address bus, data bus and control bus as shown in Fig. 5.

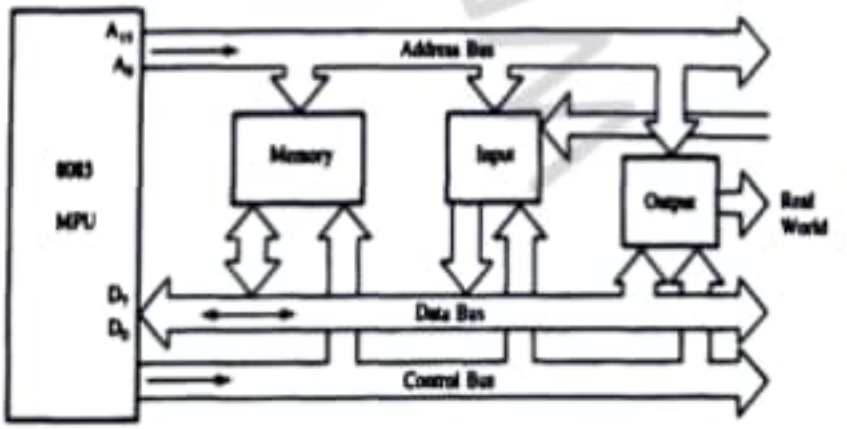



Fig. 5 The 8085 bus structure

## 3. 8085 PIN DESCRIPTION

Properties:

- It is a 8-bit microprocessor
- Manufactured with N-MOS technology
- 40 pin IC package
- It has 16-bit address bus and thus has $2^{16}$ = 64 KB addressing capability.
- Operate with 3 MHz single-phase clock
- +5 V single power supply

The logic pin layout and signal groups of the 8085nmicroprocessor are shown in Fig. 6. All the signals are classified into six groups:

- Address bus
- Data bus
- Control & status signals
- Power supply and frequency signals
- Externally initiated signals
- Serial I/O signals

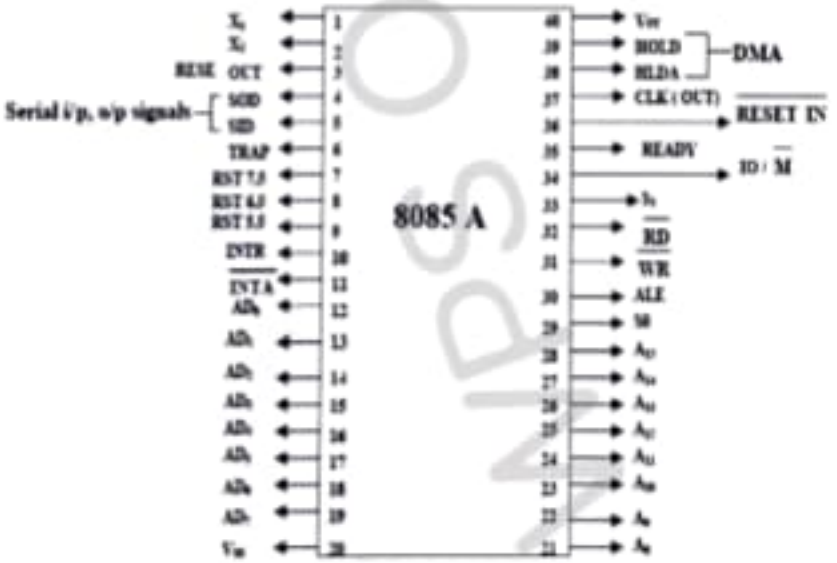



Fig. 6 8085 microprocessor pin layout and signal groups

Address and Data Buses:

- A8 – A15 (output, 3-state): Most significant eight bits of memory addresses and the eight bits of the I/O addresses. These lines enter into tri-state high impedance state during HOLD and HALT modes.
- AD0 – AD7 (input/output, 3-state): Lower significant bits of memory addresses and the eight bits of the I/O addresses during first clock cycle. Behaves as data bus

during third and fourth clock cycle. These lines enter into tri-state high impedance state during HOLD and HALT modes.

Control & Status Signals:

- ALE: Address latch enable
- RD: Read control signal.
- $\overline{WR}$ : Write control signal.
- IO/M , S1 and S0 : Status signals.

Power Supply & Clock Frequency:

- Vcc: +5 V power supply
- Vss: Ground reference
- X1, X2: A crystal having frequency of 6 MHz is connected at these two pins
- CLK: Clock output

Externally Initiated and Interrupt Signals:

- $\overline{RESET\ IN}$ : When the signal on this pin is low, the PC is set to 0, the buses are tri-stated and the processor is reset.
- RESET OUT: This signal indicates that the processor is being reset. The signal can be used to reset other devices.
- READY: When this signal is low, the processor waits for an integral number of clock cycles until it goes high.
- HOLD: This signal indicates that a peripheral like DMA (direct memory access) controller is requesting the use of address and data bus.
- HLDA: This signal acknowledges the HOLD request.
- INTR: Interrupt request is a general-purpose interrupt.
- $\overline{INTA}$ : This is used to acknowledge an interrupt.
- RST 7.5, RST 6.5, RST 5,5 – restart interrupt: These are vectored interrupts and have highest priority than INTR interrupt.
- TRAP: This is a non-maskable interrupt and has the highest priority.

Serial I/O Signals:

- SID: Serial input signal. Bit on this line is loaded to D7 bit of register A using RIM instruction.
- SOD: Serial output signal. Output SOD is set or reset by using SIM instruction.
-

## 4 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:

1. One-byte instructions: Instruction having one byte in machine code. Examples are depicted in Table 2.
I. Two-byte instructions: Instruction having two byte in machine code. Examples are depicted in Table 3
II. 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.

## 5. INSTRUCTION SET OF 8085

Data Transfer Instructions:

| Opcode | Operand | Description |
|---|---|---|
| Copy from source to destination | | |
| MOV | Rd, Rs | 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. |
| | M, Rs | |
| | Rd, M | Example: MOV B, C or MOV B, M |
| Move immediate 8-bit | | |
| MVI | Rd, 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. |
| | M, data | Example: MVI B, 57 or MVI M, 57 |
| 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 2034 or LDA XYZ |
| Load accumulator indirect | | |
| LDAX | B or Reg. pair | The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory locations 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, 2034 |
| 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 2040 |

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 4350 or STA XYZ

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 2470

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 87

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 82

## Arithmetic Instructions:

| Opcode | Operand | Description |
|---|---|---|
| Add register or memory to accumulator | | |
| ADD | R<br>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.<br>Example: ADD B or ADD M |
| Add register to accumulator with carry | | |
| ADC | R<br>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.<br>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.<br>Example: ADI 45 |
| 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.<br>Example: ACI 45 |
| 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.<br>Example: DAD H |

Subtract register or memory from accumulator

| | | |
|---|---|---|
| SUB | R<br>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.<br>Example: SUB B or SUB M |

Subtract source and borrow from accumulator

| | | |
|---|---|---|
| SBB | R<br>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.<br>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.<br>Example: SUI 45 |

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.<br>Example: SBI 45 |

Increment register or memory by 1

| | | |
|---|---|---|
| INR | R<br>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.<br>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.<br>Example: INX H |

Decrement register or memory by 1

| | | |
|---|---|---|
| DCR | R<br>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.<br>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.<br>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.<br><br>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.<br><br>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.<br><br>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.<br>Example: JMP 2034 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 2034 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 2034 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 2034 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, s=1
if (A) = (reg/mem): zero flag is set, s=0
if (A) > (reg/mem): carry and zero flags are reset, s=0
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, s=1
if (A) = data: zero flag is set, s=0
if (A) > data: carry and zero flags are reset, s=0
Example: CPI 89

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 86

Exclusive OR register or memory with accumulator

| | | |
|---|---|---|
| XRA | R<br>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.<br>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.<br>Example: XRI 86 |

Logical OR register or memory with accumulaotr

| | | |
|---|---|---|
| ORA | R<br>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.<br>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.<br>Example: ORI 86 |

Rotate accumulator left

| | | |
|---|---|---|
| RLC | none | Each binary bit of the accumulator is rotated left by one position. Bit $D_7$ is placed in the position of $D_0$ as well as in the Carry flag. CY is modified according to bit $D_7$. S, Z, P, AC are not affected.<br>Example: RLC |

Rotate accumulator right

| | | |
|---|---|---|
| RRC | none | Each binary bit of the accumulator is rotated right by one position. Bit $D_0$ is placed in the position of $D_7$ as well as in the Carry flag. CY is modified according to bit $D_0$. S, Z, P, AC are not affected.<br>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

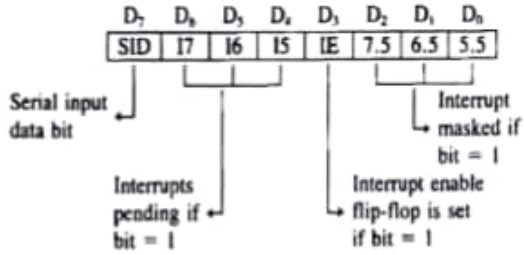
The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts (except TRAP).
Example: 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

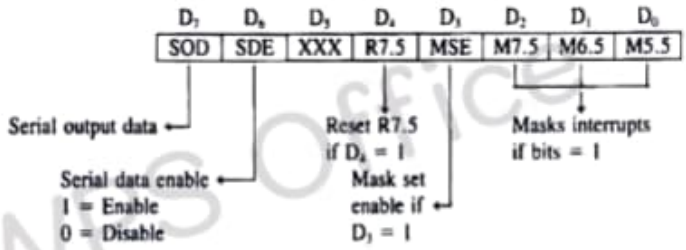



Set interrupt mask

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




- SOD — Serial Output Data: Bit $D_7$ of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit $D_6 = 1$.
- SDE — Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- XXX — Don't Care
- R7.5 — Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- MSE — Mask Set Enable: If this bit is high, it enables the functions of bits $D_2$, $D_1$, $D_0$. This is a master control over all the interrupt masking bits. If this bit is low, bits $D_2$, $D_1$, and $D_0$ do not have any effect on the masks.
- M7.5 — $D_2$ = 0, RST 7.5 is enabled.
  = 1, RST 7.5 is masked or disabled.
- M6.5 — $D_1$ = 0, RST 6.5 is enabled.
  = 1, RST 6.5 is masked or disabled.
- M5.5 — $D_0$ = 0, RST 5.5 is enabled.
  = 1, RST 5.5 is masked or disabled.

## PROGRAMING TECHNICS

The sequence of instruction called program. A set of programs written for a particular computer is known as s/w for that computer. The program is stored in RAM. The CPU takes one instruction of the program at a time from the RAM and executes it. It executes all the instruction of the program one by one to produce the desired result. A computer used the binary digits for its operation. Hence the instructions are coded and stored in the memory in the form of zeroes and ones. The program written in the form of 0's and 1's are called machine language programs.

**PROGRAMS:-**

Write a program to find 1's complement of an 8-bit number. 96H = 1001 0110
1's complement = 0110 1001 = 69H

The number is placed in the memory location 2501H The result is stored in memory location 2502H. DATA

2501 96H

Result

2502 69H

Write a program to find 2's complement of an 8-bit number. 96 = 1001 0110
1's complement = 0110 1001
2's complement = 0110 1010 = 6A

The number is placed in memory location 2501H The result is to be stored in memory location 2502H

| Address | Machine code | Mnemonics | Operands | Comments |
|---|---|---|---|---|
| 2000 | 3A,01,25 | LDA | 2501H | Get data in accumulator |
| 2003 | 2F | CMA | | Take its 1's complement |
| 2004 | 3C | INR | A | Take 2's complement |
| 2005 | 32, 02, 25 | STA | 2502H | Store result in 2502H |
| 2008 | 76 | HLT | | Stop |

DATA

2501 96H

Result

2502 6AH

Write a program to Mask off least significant 4-bit of an 8-bit number. Number = A6 = 1010 0110

Result = A0 = 1010 0000

| Address | Machine code | Mnemonics | Operands | Comments |
|---|---|---|---|---|
| 2000 | 3A,01,25 | LDA | 2501H | Get data in accumulator |
| 2003 | E6,F0 | ANI | F0 | Mask off the least significant 4bits |
| 2005 | 32,02,25 | STA | 2502H | Store result in 2502H |
| 2008 | 76 | HLT | | stop |

DATA 2501 A6

Result 2502 A0

Write a program to Mask off most significant 4-bit of an 8-bit number. Number = A6 = 1010 0110

Result = 06 = 0000 0110

| Address | Machine code | Mnemonics | Operands | Comments |
|---|---|---|---|---|
| 2000 | 3A,01,25 | LDA | 2501H | Get data in accumulator |
| 2003 | E6,F0 | ANI | 0F | Mask off the most significant 4bits |
| 2005 | 32,02,25 | STA | 2502H | Store result in 2502H |
| 2008 | 76 | HLT | | stop |

DATA 2501 A6

Result 2502 06



Write a program to find the largest number in a data array. Number = 98, 75 and 99

| Address | Machine Code | Labels | Mnemonics | Operands | Comments |
|---|---|---|---|---|---|
| 2000 | 21,00,25 | | LXI | H,2500H | Address for count in H-L pair |
| 2003 | 4E | | MOV | C,M | Count in register C |
| 2004 | 23 | | INX | H | Address of 1st number in H-L pair |
| 2005 | 7E | | MOV | A,M | 1st Number in accumulator |
| 2006 | 0D | | DCR | C | Decrement count |
| 2007 | 23 | LOOP | INX | H | Address of next number |
| 2008 | BE | | CMP | M | compare next number with previous maximum. Is next number > previous maximum |

| | | | | | |
|---|---|---|---|---|---|
| 2009 | D2,0D,20 | | JNC | AHEAD | No, larger number is in accumulator. Go to the lable AHEAD. |
| 200C | 7E | | MOV | A,M | Yes, get larger number in accumulator |
| 200D | 0D | AHEAD | DCR | C | Decrement count |
| 200E | C2,07,20 | | JNZ | LOOP | |
| 2011 | 32,50,24 | | STA | 2450H | Store result in 2450H |
| 2014 | 76 | | HLT | | Stop. |

DATA 2500 03

2501-98

2502-75

2503-99

Result 2450-99

Write a program to find the smallest number in a data array. The numbers of a series are: 86,58 and 75.

| Address | Machine Code | Labels | Mnemonics | Operands | Comments |
|---|---|---|---|---|---|
| 2000 | 21,00,25 | | LXI | H,2500H | Address for count in H-L pair |
| 2003 | 4E | | MOV | C,M | Count in register C |
| 2004 | 23 | | INX | H | Address of 1st number in H-L pair |
| 2005 | 7E | | MOV | A,M | 1st Number in accumulator |
| 2006 | 0D | | DCR | C | Decrement count |
| 2007 | 23 | LOOP | INX | H | Address of next number in H-L pair |
| 2008 | BE | | CMP | M | compare next number with previous smallest. Is previous smallest less then next number |
| 2009 | DA,0D,20 | | JC | AHEAD | Yes, smallest number in accumulator. Go to the AHEAD. |
| 200C | 7E | | MOV | A,M | NO, get next number in accumulator |

| | | | | | |
|---|---|---|---|---|---|
| 200D | 0D | AHEAD | DCR | C | Decrement count |
| 200E | C2,07,20 | | JNZ | LOOP | |
| 2011 | 32,50,24 | | STA | 2450H | Store smallest number in 2450H |
| 2014 | 76 | | HLT | | Stop. |

DATA
2500 03H
2501-86H
2502-58H
2503-75H
Result 2450 58H

Write a program to find the smaller of two number. The numbers are 84H and 99H

| Address | Machine Code | Labels | Mnemonics | Operands | Comments |
|---|---|---|---|---|---|
| 2000 | 21,01,25 | | LXI | H,2501H | Address of 1st number in H-L pair |
| 2003 | 7E | | MOV | A,M | 1ST number in accumulator |
| 2004 | 23 | | INX | H | Address of 2nd number in H-L pair |
| 2005 | BE | | CMP | M | Compare 2nd number with 1st.Is 1st number < 2nd number. |
| 2006 | DA,0A,20 | | JC | AHEAD | Yes, smaller number is in accumulator. Go to AHEAD. |
| 2009 | 7E | | MOV | A,M | No, get 2nd number in accumulator. |
| 200A | 32,03,25 | AHEAD | STA | 2503H | Store smaller number in 2503H |
| 200D | 76 | | | HLT | stop. |

DATA
2501 84H
2502—99H
Result:
2503 84H

Write a program to find the larger of two numbers. The numbers are 98H and 87H.

DATA

2501 98H

2502—87H

| Address | Machine Code | Labels | Mnemonics | Operands | Comments |
|---|---|---|---|---|---|
| 2000 | 21,01,25 | | LXI | H,2501H | Address of 1st number in H-L pair |
| 2003 | 7E | | MOV | A,M | 1ST number in accumulator |
| 2004 | 23 | | INX | H | Address of 2nd number in H-L pair |
| 2005 | BE | | CMP | M | Compare 2nd number with 1st.Is 2nd number > 1st number. |
| 2006 | D2,0A,20 | | JNC | AHEAD | No, larger number is in accumulator. Go to AHEAD. |
| 2009 | 7E | | MOV | A,M | Yes, get 2nd number in accumulator. |
| 200A | 32,03,25 | AHEAD | STA | 2503H | Store larger number in 2503H |
| 200D | 76 | | HLT | | stop. |

Result:

2503 98H



Write program to move a block of data from one section of memory to another section of memory.

| Address | Machine Code | Mnemonics | Comments |
|---|---|---|---|
| 2400 | 21,00,20 | LXI H,2000H | Get memory address of count. |
| 2403 | 4E | MOV C,M | Count in register C. |
| 2404 | 23 | INX H | Source address of data in H-L pair. |
| 2405 | 11,01,22 | LXI D 2201 | Destiny address of data in D-E pair. |
| 2408 | 7E LOOP | MOV A,M | Data from source address to ACC. |
| 2409 | EB | XCHG | Destiny address in H-L pair. |

| | | | |
|---|---|---|---|
| 240A | 77 | MOV M,A | Data to Destiny address . |
| 240B | EB | XCHG | Source address of data in H-L pair |
| 240C | 23 | INX H | Source address of next data. |
| 240D | 13 | INX D | Destiny address of next data. |
| 240E | 0D | DCR C | Decrement count. |
| 240F | C2,08,24 | JNZ LOOP | Jump to label LOOP. |
| 2412 | 76 | HLT | Stop. |

DATA:
2000—05
2001—01
2002---02
2003---03
2004---04
2005---05

RESULT:
2201---01
2202---02
2203---03
2204---04
2205---05

**SUBROUTINES:-**

A subroutine is a group of instructions written separately from the main program to perform a function that occurs repeatedly in main program. Instead of writing a subprogram repeatedly in main program, simply write it once in another location. Whenever needed by the main program we can call it. This technique overcomes the repetition of a subprogram in main program.

CALL RETURN structure of subroutine.

## 6. 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 in Fig. 7.

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. IO/$\overline{M}$ 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 $\overline{RD}$ 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 $\overline{RD}$ 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.

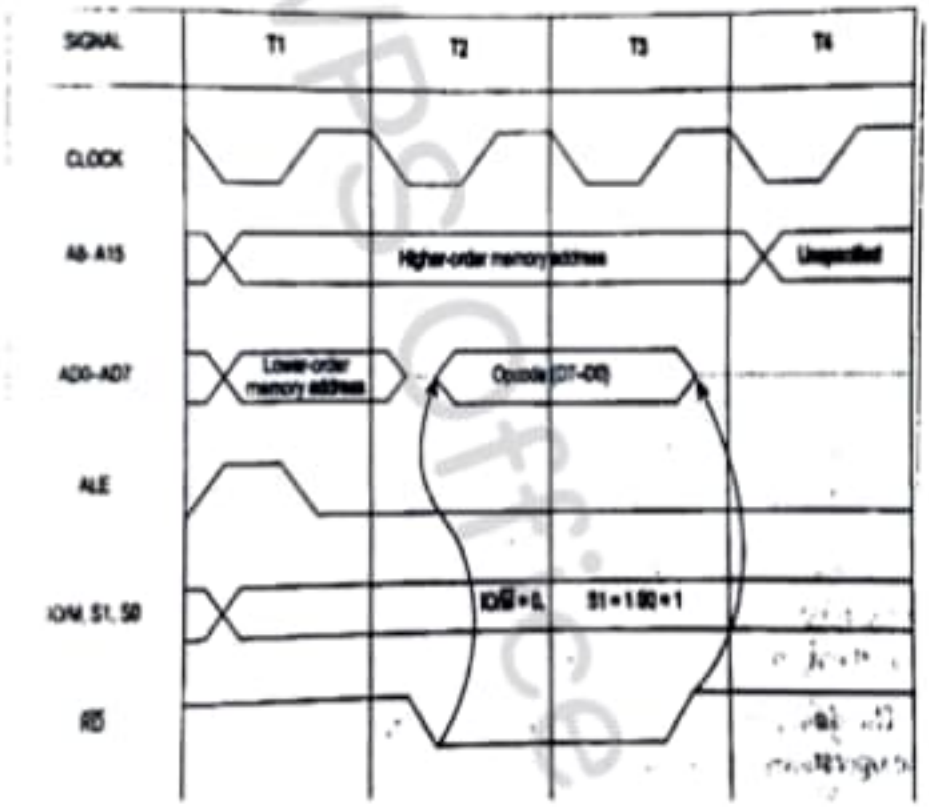



Fig. 7 Timing diagram for opcode fetch cycle

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. 8.

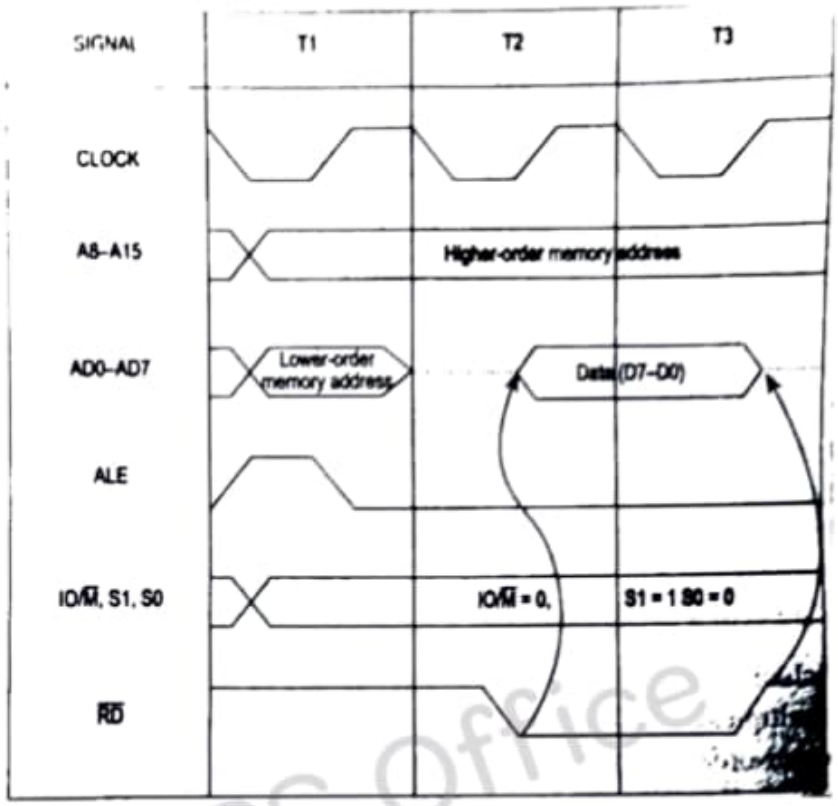



Fig. 8 Timing diagram for memory read machine cycle



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 $\overline{WR}$ signal is made low. The timing diagram of this cycle is given in Fig. 9.

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 in Fig. 10.

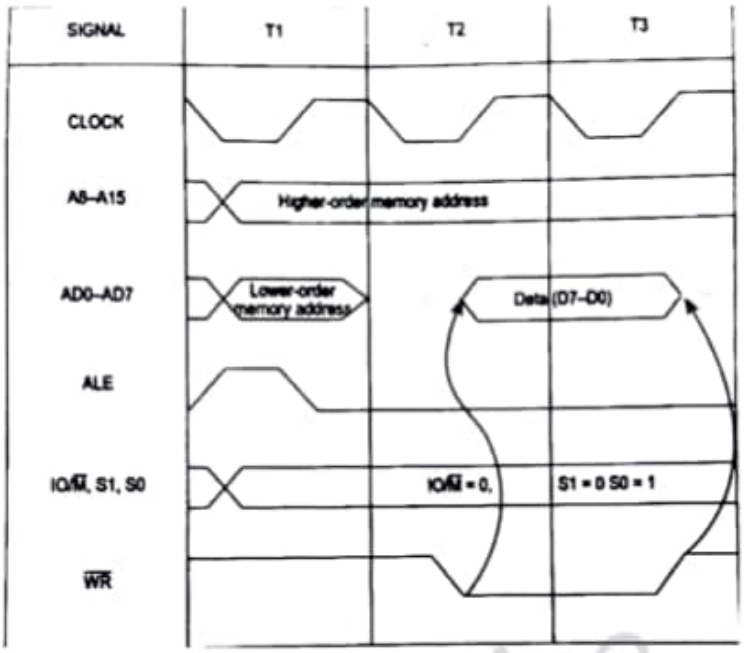



Fig. 9 Timing diagram for memory write machine cycle

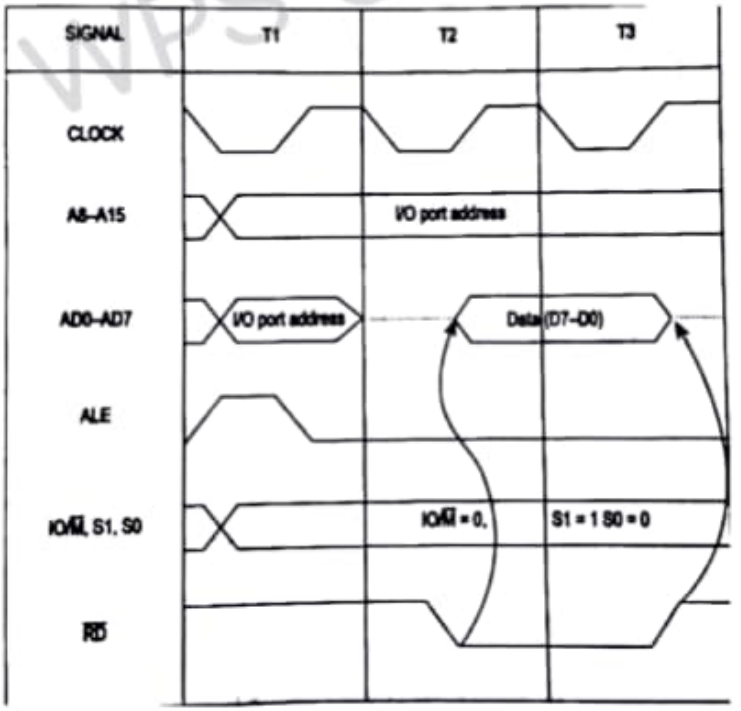



Fig. 10 Timing diagram I/O read machine cycle

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 Fig. 11

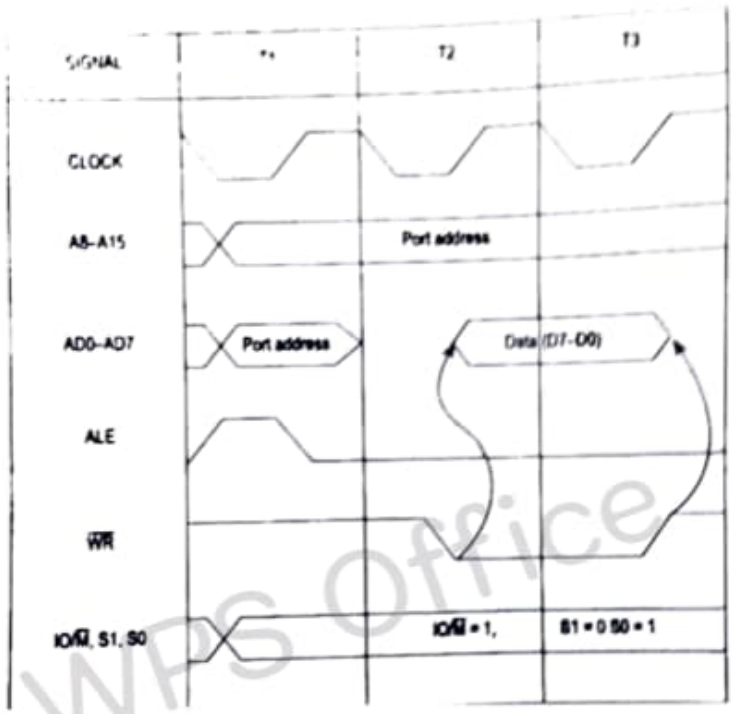



Fig. 11 Timing diagram I/O write machine cycle

Ex: Timing diagram for IN 80H.

The instruction and the corresponding codes and memory locations are given in Table 5.

Table 5 IN instruction

| Address | Mnemonics | Opcode |
|---|---|---|
| 800F | IN 80H | DB |
| 8010 | | 80 |

i. During the first machine cycle, the opcode DB is fetched from the memory, placed in the instruction register and decoded.
ii. During second machine cycle, the port address 80H is read from the next memory location.
iii. During the third machine cycle, the address 80H is placed in the address bus and the data read from that port address is placed in the accumulator.

The timing diagram is shown in Fig. 12.

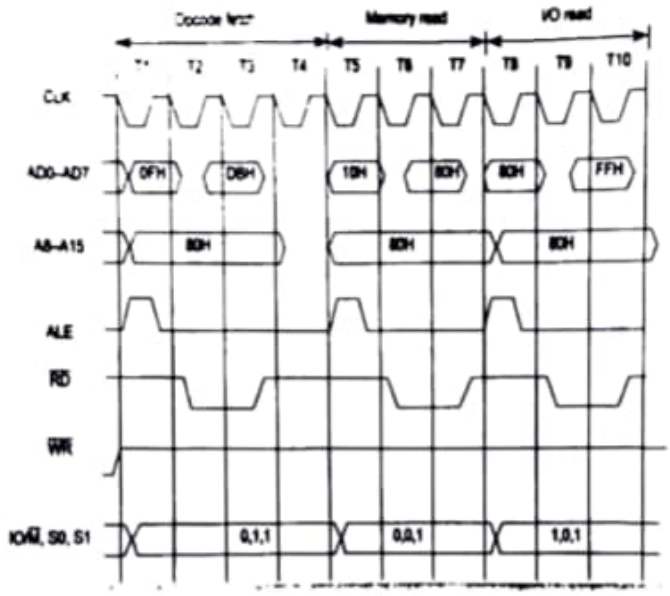
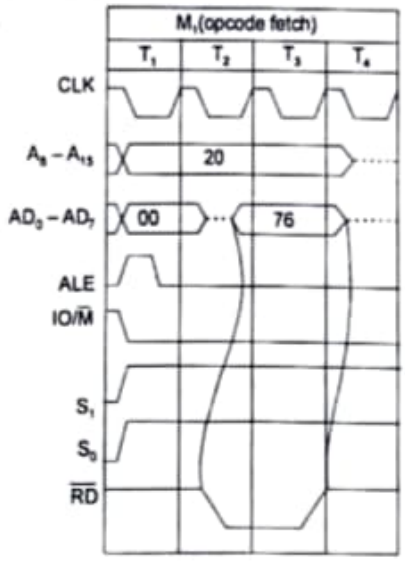



Fig. 12 Timing diagram for the IN instruction

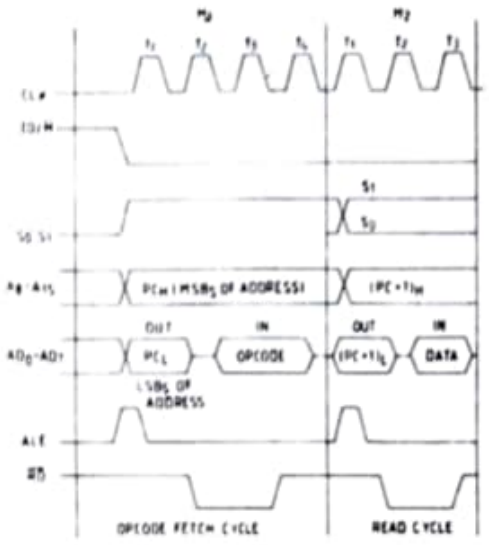


## 1. Timing Diagram for MOV A, B.



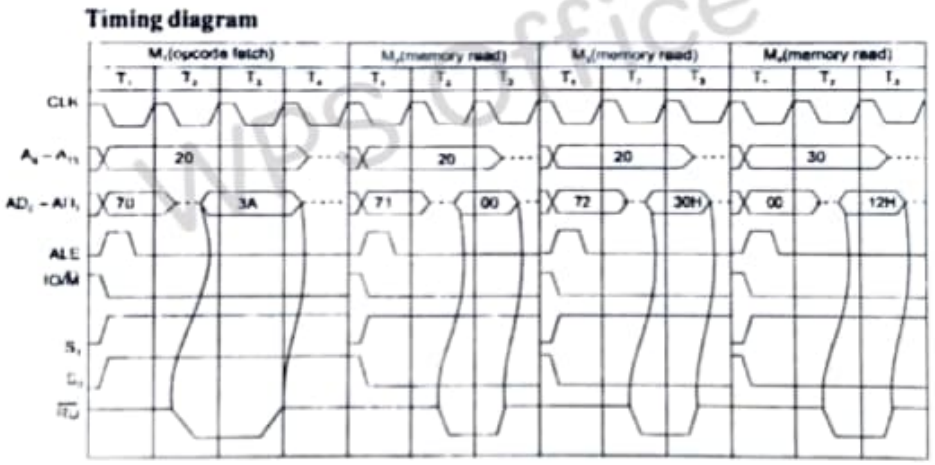

*Timing diagram for MOV A, B*

2. Timing Diagram for MVI r, Data.



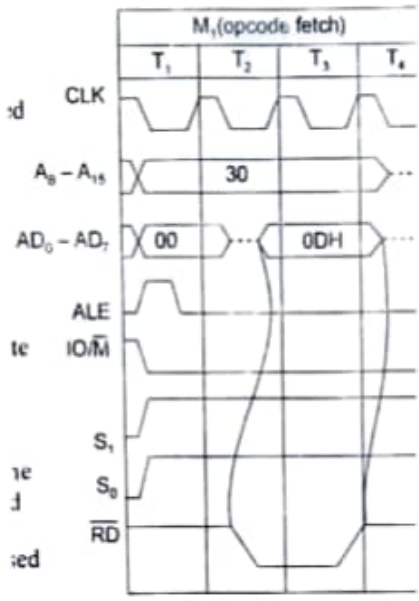

Timing Diagram for MVI r, Data

3. Timing Diagram for LDA 3000H.



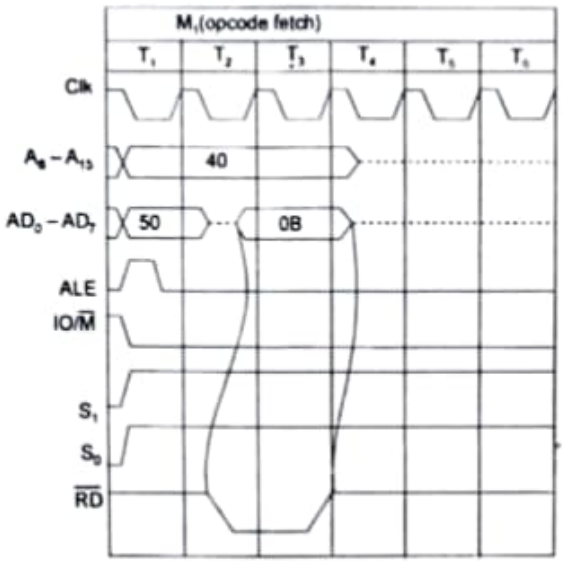

*Timing diagram for LDA 3000H*

4. Timing Diagram for DCR C.




*Timing diagram for DC*

5. Timing Diagram for DCX B.




*Timing diagram for DCX B*



## 7. 8085 INTERRUPTS

Interrupt Structure:

Interrupt is the mechanism by which the processor is made to transfer control from its current program execution to another program having higher priority. The interrupt signal may be given to the processor by any external peripheral device.

The program or the routine that is executed upon interrupt is called interrupt service routine (ISR). After execution of ISR, the processor must return to the interrupted program. Key features in the interrupt structure of any microprocessor are as follows:

i. Number and types of interrupt signals available.
ii. The address of the memory where the ISR is located for a particular interrupt signal. This address is called interrupt vector address (IVA).
iii. Masking and unmasking feature of the interrupt signals.
iv. Priority among the interrupts.
v. Timing of the interrupt signals.
vi. Handling and storing of information about the interrupt program (status information).

Types of Interrupts:

Interrupts are classified based on their maskability, IVA and source. They are classified as:

i. Vectored and Non-Vectored Interrupts
   - Vectored interrupts require the IVA to be supplied by the external device that gives the interrupt signal. This technique is vectoring, is implemented in number of ways.
   - Non-vectored interrupts have fixed IVA for ISRs of different interrupt signals.

ii. Maskable and Non-Maskable Interrupts
   - Maskable interrupts are interrupts that can be blocked. Masking can be done by software or hardware means.
   - Non-maskable interrupts are interrupts that are always recognized; the corresponding ISRs are executed.

iii. Software and Hardware Interrupts
   - Software interrupts are special instructions, after execution transfer the control to predefined ISR.
   - Hardware interrupts are signals given to the processor, for recognition as an interrupt and execution of the corresponding ISR.

Interrupt Handling Procedure:

The following sequence of operations takes place when an interrupt signal is recognized:

i. Save the PC content and information about current state (flags, registers etc) in the stack.
ii. Load PC with the beginning address of an ISR and start to execute it.
iii. Finish ISR when the return instruction is executed.
iv. Return to the point in the interrupted program where execution was interrupted.



Interrupt Sources and Vector Addresses in 8085:

Software Interrupts:

8085 instruction set includes eight software interrupt instructions called Restart (RST) instructions. These are one byte instructions that make the processor execute a subroutine at predefined locations. Instructions and their vector addresses are given in Table 6.

Table 6 Software interrupts and their vector addresses

| Instruction | Machine hex code | Interrupt Vector Address |
|---|---|---|
| RST 0 | C7 | 0000H |
| RST 1 | CF | 0008H |
| RST 2 | D7 | 0010H |
| RST 3 | DF | 0018H |
| RST 4 | E7 | 0020H |
| RST 5 | EF | 0028H |
| RST 6 | F7 | 0030H |
| RST 7 | FF | 0032H |

The software interrupts can be treated as CALL instructions with default call locations. The concept of priority does not apply to software interrupts as they are inserted into the program as instructions by the programmer and executed by the processor when the respective program lines are read.

Hardware Interrupts and Priorities:

8085 have five hardware interrupts – INTR, RST 5.5, RST 6.5, RST 7.5 and TRAP. Their IVA and priorities are given in Table 7.

Table 7 Hardware interrupts of 8085

| Interrupt | Interrupt vector address | Maskable or non-maskable | Edge or level triggered | priority |
|---|---|---|---|---|
| TRAP | 0024H | Non-makable | Level | 1 |
| RST 7.5 | 003CH | Maskable | Rising edge | 2 |
| RST 6.5 | 0034H | Maskable | Level | 3 |
| RST 5.5 | 002CH | Maskable | Level | 4 |
| INTR | Decided by hardware | Maskable | Level | 5 |

Masking of Interrupts:

Masking can be done for four hardware interrupts INTR, RST 5.5, RST 6.5, and RST 7.5. The masking of 8085 interrupts is done at different levels. Fig. 13 shows the organization of hardware interrupts in the 8085.



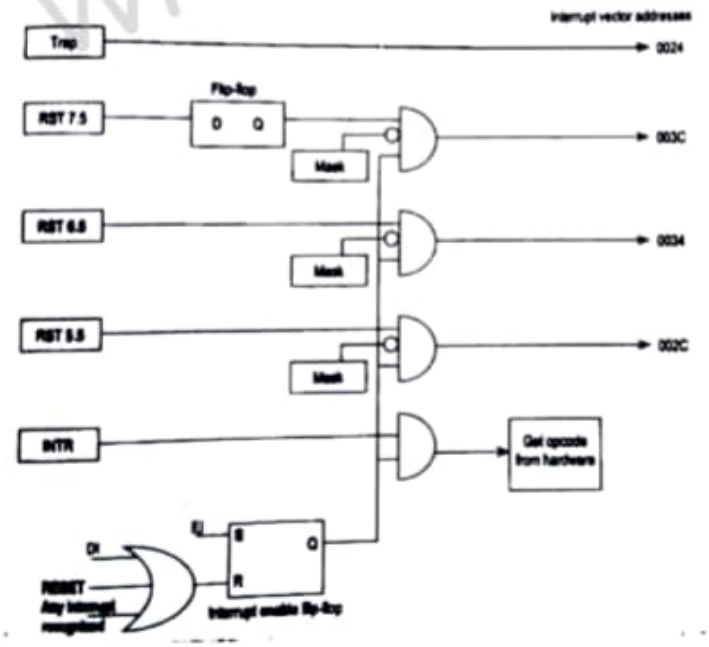



Fig. 13 Interrupt structure of 8085

The Fig. 13 is explained by the following five points:

i. The maskable interrupts are by default masked by the Reset signal. So no interrupt is recognized by the hardware reset.
ii. The interrupts can be enabled by the EI instruction.
iii. The three RST interrupts can be selectively masked by loading the appropriate word in the accumulator and executing SIM instruction. This is called software masking.
iv. All maskable interrupts are disabled whenever an interrupt is recognized.
v. All maskable interrupts can be disabled by executing the DI instruction.

RST 7.5 alone has a flip-flop to recognize edge transition. The DI instruction reset interrupt enable flip-flop in the processor and the interrupts are disabled. To enable interrupts, EI instruction has to be executed.

SIM Instruction:

The SIM instruction is used to mask or unmask RST hardware interrupts. When executed, the SIM instruction reads the content of accumulator and accordingly mask or unmask the interrupts. The format of control word to be stored in the accumulator before executing SIM instruction is as shown in Fig. 14.

| Bit position | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| Name | SOD | SDE | X | R7.5 | MSE | M7.5 | M6.5 | M5.5 |
| Explanation | Serial data to be sent | Serial data enable— set to 1 for sending | Not used | Reset RST 7.5 flip-flop | Mask set enable— Set to 1 to mask interrupts | Set to 1 to mask RST 7.5 | Set to 1 to mask RST 6.5 | Set to 1 to mask RST 5.5 |



Fig. 14 Accumulator bit pattern for SIM instruction

In addition to masking interrupts, SIM instruction can be used to send serial data on the SOD line of the processor. The data to be send is placed in the MSB bit of the accumulator and the serial data output is enabled by making D6 bit to 1.

RIM Instruction:

RIM instruction is used to read the status of the interrupt mask bits. When RIM instruction is executed, the accumulator is loaded with the current status of the interrupt masks and the pending interrupts. The format and the meaning of the data stored in the accumulator after execution of RIM instruction is shown in Fig. 15.

In addition RIM instruction is also used to read the serial data on the SID pin of the processor. The data on the SID pin is stored in the MSB of the accumulator after the execution of the RIM instruction.

| Bit position | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| Name | SID | I7.5 | I6.5 | I5.5 | IE | M7.5 | M6.5 | M5.5 |
| Explanation | Serial input data in the SID pin | Set to 1 if RST 7.5 is pending | Set to 1 if RST 6.5 is pending | Set to 1 if RST 5.5 is pending | Set to 1 if interrupts are enabled | Set to 1 if RST 7.5 is masked | Set to 1 if RST 6.5 is masked | Set to 1 if RST 5.5 is masked |

Fig. 15 Accumulator bit pattern after execution of RIM instruction

Ex: Write an assembly language program to enables all the interrupts in 8085 after reset.

EI : Enable interrupts

MVI A, 08H : Unmask the interrupts

SIM : Set the mask and unmask using SIM instruction

Timing of Interrupts:

The interrupts are sensed by the processor one cycle before the end of execution of each instruction. An interrupts signal must be applied long enough for it to be recognized. The longest instruction of the 8085 takes 18 clock periods. So, the interrupt signal must be applied for at least 17.5 clock periods. This decides the minimum pulse width for the interrupt signal.

The maximum pulse width for the interrupt signal is decided by the condition that the interrupt signal must not be recognized once again. This is under the control of the programmer.

**Development of microprocessors (Visible)**

Microprocessors have undergone significant evolution over the past four decades. This development is clearly perceptible to a common user, especially, in terms of phenomenal growth in capabilities of personal computers. Development of some of the microprocessors can be given as follows.

| | | |
|---|---|---|
| Intel 4004 | 4 bit (2300 PMOS transistors) | 1971 |
| Intel 8080<br>8085 | 8 bit (NMOS)<br>8 bit | 1974 |
| Intel 8088<br>8086 | 16 bit<br>16 bit | 1978 |
| Intel 80186<br>80286 | 16 bit<br>16 bit | 1982 |
| Intel 80386 | 32 bit (275000 transistors) | 1985 |
| Intel 80486 SX<br>DX | 32 bit<br>32 bit (built in floating point unit) | 1989 |
| Intel 80586 I<br>MMX<br>Celeron<br>II<br>III<br>IV | 64 bit | 1993<br>1997<br>1999<br>2000 |
| Z-80 (Zilog) | 8 bit | 1976 |
| Motorola Power PC<br>601<br>602<br>603 | 32-bit | 1993<br>1995 |



We use more number of microcontrollers compared to microprocessors. Microprocessors are primarily used for computational purpose, whereas microcontrollers find wide application in devices needing real time processing / control.

Application of microcontrollers are numerous. Starting from domestic applications such as in washing machines, TVs, airconditioners, microcontrollers are used in automobiles, process control industries , cell phones, electrical drives, robotics and in space applications.

**Microcontroller Chips**

Broad Classification of different microcontroller chips could be as follows:

- Embedded (Self -Contained) 8 - bit Microcontroller
- 16 to 32 Microcontrollers
- Digital Signal Processors

## 8051 microcontroller

**What is a Microcontroller?**

A Microcontroller is a programmable digital processor with necessary peripherals. Both microcontrollers and microprocessors are complex sequential digital circuits meant to carry out job according to the program / instructions. Sometimes analog input/output interface makes a part of microcontroller circuit of mixed mode(both analog and digital nature).

**Microcontrollers Vs Microprocessors**

1. A microprocessor requires an external memory for program/data storage. Instruction execution requires movement of data from the external memory to the microprocessor or vice versa. Usually, microprocessors have good computing power and they have higher clock speed to facilitate faster computation.
2. A microcontroller has required on-chip memory with associated peripherals. A microcontroller can be thought of a microprocessor with inbuilt peripherals.
3. A microcontroller does not require much additional interfacing ICs for operation and it functions as a stand alone system. The operation of a microcontroller is multipurpose, just like a Swiss knife.
4. Microcontrollers are also called embedded controllers. A microcontroller clock speed is limited only to a few tens of MHz. Microcontrollers are numerous and many of them are application specific.

**Development/Classification of microcontrollers (Invisible)**

Microcontrollers have gone through a silent evolution (invisible). The evolution can be rightly termed as silent as the impact or application of a microcontroller is not well known to a common user, although microcontroller technology has undergone significant change since early 1970's. Development of some popular microcontrollers is given as follows.

| | | |
|---|---|---|
| Intel 4004 | 4 bit (2300 PMOS trans, 108 kHz) | 1971 |
| Intel 8048 | 8 bit | 1976 |
| Intel 8031 | 8 bit (ROM-less) | . |
| Intel 8051 | 8 bit (Mask ROM) | 1980 |
| Microchip PIC16C64 | 8 bit | 1985 |
| Motorola 68HC11 | 8 bit (on chip ADC) | . |
| Intel 80C196 | 16 bit | 1982 |
| Atmel AT89C51 | 8 bit (Flash memory) | . |
| Microchip PIC 16F877 | 8 bit (Flash memory + ADC) | . |

The control outputs produced by the 8288 are DEN, DT/R and ALE. These 3 signals provide the same functions as those described for the minimum system mode. This set of bus commands and control signals is compatible with the Multibus and industry standard for interfacing microprocessor systems.

**8289 Bus Arbiter – Bus Arbitration and Lock Signals:**

This device permits processors to reside on the system bus. It does this by implementing the Multibus arbitration protocol in an 8086-based system. Addition of the 8288 bus controller and 8289 bus arbiter frees a number of the 8086 pins for use to produce control signals that are needed to support multiple processors. Bus priority lock (LOCK) is one of these signals. It is input to the bus arbiter together with status signals S0 through S2.

**Queue Status Signals:** Two new signals that are produced by the 8086 in the maximum-mode system are queue status outputs QS0 and QS1. Together they form a 2-bit queue status code, QS1QS0. Following table shows the four different queue status.

| $QS_1$ | $QS_0$ | Queue Status |
|---|---|---|
| 0 (low) | 0 | No Operation. During the last clock cycle, nothing was taken from the queue. |
| 0 | 1 | First Byte. The byte taken from the queue was the first byte of the instruction. |
| 1 (high) | 0 | Queue Empty. The queue has been reinitialized as a result of the execution of a transfer instruction. |
| 1 | 1 | Subsequent Byte. The byte taken from the queue was a subsequent byte of the instruction. |

Queue status codes