SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF SCIENCE AND HUMANITIES

DEPARTMENT OF PHYSICS

UNIT - 1

Microprocessor and Microcomputer – SPH1313

Introduction

Introduction to Microprocessor

Definition:

- "The microprocessor is a multipurpose, clock driven, register based, digital-integrated circuit which accepts binary data as input, processes it according to instructions stored in its memory, and provides results as output."
- "Microprocessor is a computer Central Processing Unit (CPU) on a single chip that contains millions of transistors connected by wires."

Introduction:

- A microprocessor is designed to perform arithmetic and logic operations that make use of small number-holding areas called registers.
- Typical microprocessor operations include adding, subtracting, comparing two numbers, and fetching numbers from one area to another.

Components of Microprocessor

- Microprocessor is capable of performing various computing functions andmaking decisions to change the sequence of program execution.
- The microprocessor can be divided into three segments as shown in the figure, Arithmetic/logic unit (ALU), register array, and control unit.
- These three segment is responsible for all processing done in a computer

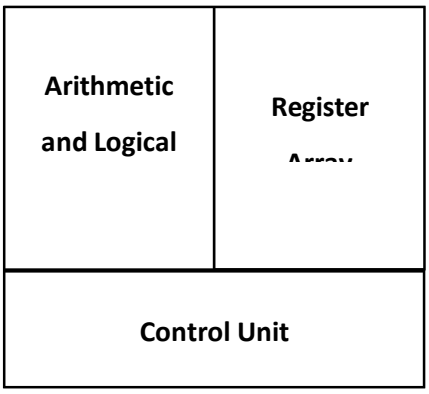


Figure: Components of Microprocessor

Arithmetic and logic unit (ALU)

- It is the unit of microprocessor where various computing functions are performed on the data.
- It performs arithmetic operations such as addition, subtraction, and logical operations such as OR,AND, and Exclusive-OR.
- It is also known as the brain of the computer system.

Register array

- It is the part of the register in microprocessor which consists of various registers identified by letters such as B, C, D, E, H, and L.a
- Registers are the small additional memory location which are used to store and transfer data and programs that are currently being executed.

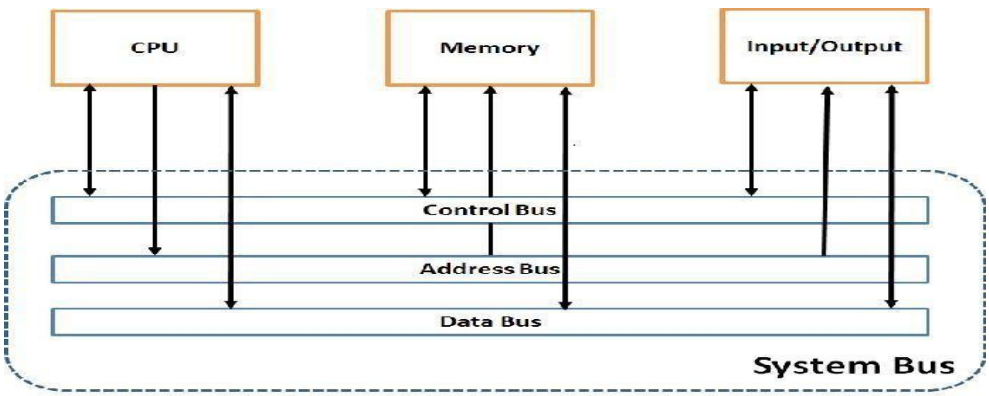
Control unit

- The control unit provides the necessary timing and control signals to all the operations in the microcomputer.
- It controls and executes the flow of data between the microprocessor, memory and peripherals.
- The control bus is bidirectional and assists the CPU in synchronizing control signals to internal devices and external components.
- This signal permits the CPU to receive or transmit data from main memory.

System bus (data, address an control bus)

This network of wires or electronic pathways is called the 'Bus'.

- A system bus is a single computer bus that connects the major components of a computer system.



Address Bus

- It is a group of wires or lines that are used to transfer the addresses of Memory or I/O devices.
- It is unidirectional.
- The width of the address bus corresponds to the maximum addressing capacity of the bus, or the largest address within memory that the bus can work with.
- The addresses are transferred in binary format, with each line of the address bus carrying a single binary digit.
- Therefore the maximum address capacity is equal to two to the power of the number of lines present (2^lines).

Data Bus

- It is used to transfer data within Microprocessor and Memory/Input or Outputdevices.
- It is bidirectional as Microprocessor requires to send or receive data.
- Each wire is used for the transfer of signals corresponding to a single bit of binary data.
- As such, a greater width allows greater amounts of data to be transferred at the same time.

Control Bus

- Microprocessor uses control bus to process data,i.e. what to do with the selected memory location.
- Some control signals are Read, Write and Opcode fetch etc.
- Various operations are performed by microprocessor with the help of control bus.
- This is a dedicated bus, because all timing signals are generated according to control signal.

Microprocessor systems with bus organization

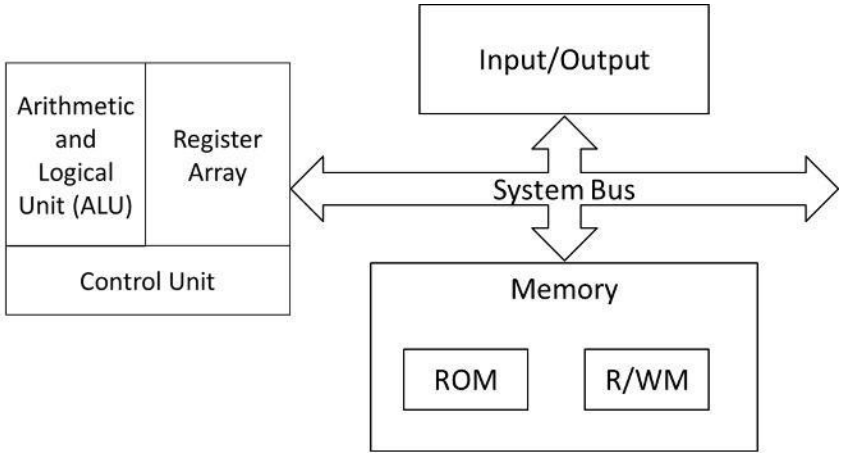


Figure: Microprocessor systems with bus organization

- To design any meaningful application microprocessor requires support of other auxiliary devices.
- In most simplified form a microprocessor based system consist of a microprocessor, I/O (input/output) devices and memory.
- These components are interfaced (connected) with microprocessor over a common communication path called system bus. Typical structure of a microprocessor based system is shown in Figure.
- Here, microprocessor is master of the system and responsible for executing the program and coordinating with connected peripherals as required.
- Memory is responsible for storing program as well as data. System generally consists of two types of memories ROM (Read only and non-volatile) and RAM (Read/Write and volatile).
- I/O devices are used to communicate with the environment. Keyboard can be example of input devices and LED, LCD or monitor can be example of output device.
- Depending on the application level of sophistication varies in a microprocessor based systems. For example: washing machine, computer.

Internal Architecture of 8085

1. Features of 8085 microprocessor.

- It is an 8 bit microprocessor.
- It is manufactured with N-MOS technology.
- It has 16-bit address bus and hence can address up to 216 = 65536 bytes (64KB) memory locations through A0-A15
- The first 8 lines of address bus and 8 lines of data bus are multiplexed AD0 – AD7
- Data bus is a group of 8 lines D0 – D7
- It supports external interrupt request. .
- A 16 bit program counters (PC)
- A 16 bit stack pointer (SP)
- Six 8-bit general purpose register arranged in pairs: BC, DE, HL.
- It requires a signal +5V power supply and operates at 3.2 MHZ single phase clock.
- It is enclosed with 40 pins DIP (Dual in line package).

Explain 8085 microprocessor architecture.

The 8085 microprocessor consists of the following functional units −

1. **Register Array**
2. **ALU and associated circuitry**
3. **Instruction register and Decoder**
4. **Timing and Control Unit**

2. Interrupt and Serial I/O

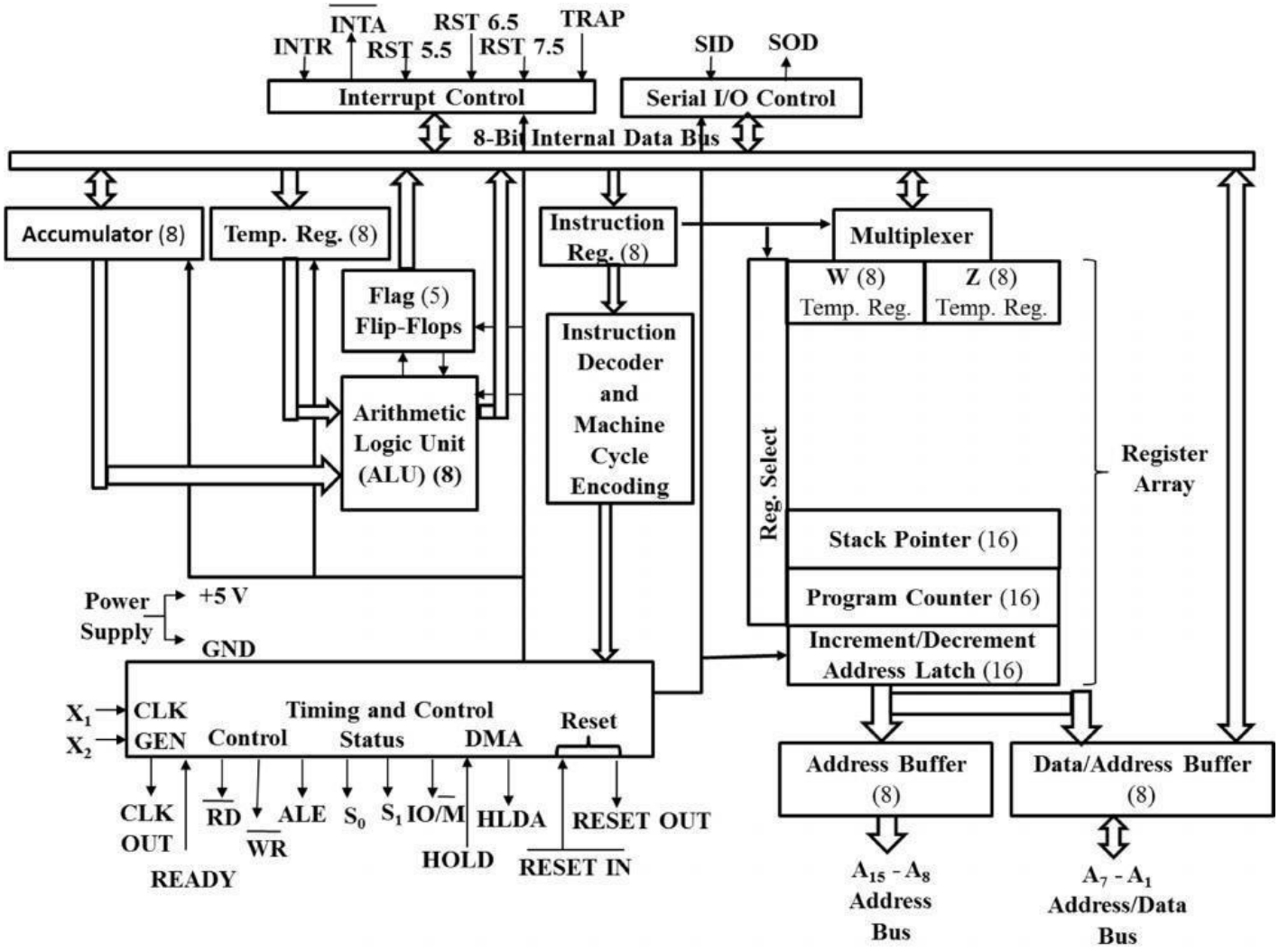


Figure: 8085 microprocessor architecture.

1.REGISTER ARRAY

General purpose register

There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8- bit data ranging from 00H to FF H (0-255)

These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

. Each register can hold 16-bit data ranging from 0000 $_H$ to FFFF $_H$(0-65,535)

The user can use these registers to store or copy a data temporarily during the execution of a program by using data transfer instructions.

Program counter (PC)

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

Stack pointer(SP)

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

Temporary register

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

Address buffer and address-data buffer register

The content stored in the stack pointer and program counter is loaded into the address buffer and address- data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.

Address bus and data bus

Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address of memory or I/O devices.

2. ALU AND ASSOCIATED CIRCUITRY

Arithmetic and logic unit

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

Accumulator

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU. The result of an operation is stored in the accumulator. It is also identified as register A.

Flag register

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops −

- Sign (S)
- Zero (Z)
- Auxiliary Carry (AC)
- Parity (P)
- Carry (C)

Its bit position is shown in the following table −

D 7	**D 6**	**D5**	**D 4**	**D3**	**D 2**	**D1**	**D 0**
S	Z	x	A C	x	P	x	C Y

3. INSTRUCTION REGISTER AND DECODER

It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

4. TIMING AND CONTROL UNIT

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits −

- Control Signals: READY, RD', WR', ALE
- Status Signals: S0, S1, IO/M'
- DMA Signals: HOLD, HLDA

- RESET Signals: RESET IN, RESET OUT

5. INTERRUPT AND SERIAL I/O

Interrupt control

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

Serial Input/output control

It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

Explain Flags Registers in 8085

- Flag register includes five flip-flops, which are set or reset after an operation according to the data conditions 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.
- The microprocessor uses these flags to set and test data conditions.

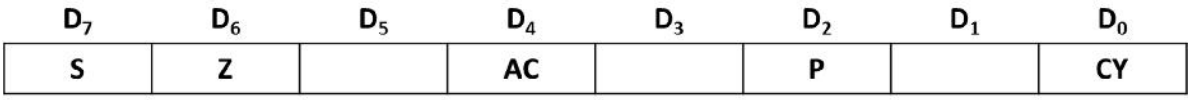


Figure: Flags registers in 8085.

- The flags are stored in the 8-bit register so that the programmer can examine these flags by accessing the register through an instruction.
- These flags have critical importance in the decision-making process of the microprocessor.
- The conditions (set or reset) of the flags are tested through the software instructions.
- For instance, JC (jump on carry) is implemented to change the sequence of a program when CY flag is set.

Z (Zero) Flag:

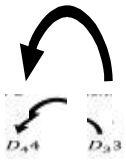
- This flag indicates whether the result of mathematical or logical operation is zero or not.
- If the result of the current operation is zero, then this flag will be set, otherwise reset.

CY (Carry) Flag:

- This flag indicates, whether, during an addition or subtraction operation, carry or borrow is generated or not, if generated then this flag bit will beset.

AC (Auxiliary Carry) Flag:

- It shows carry propagation from D3 position to D4 position.



1	0	0	0	1	1	0	0
0	0	1	0	1	0	1	1
1	0	1	1	0	1	1	1

Figure: Auxiliary Carry.

- As shown in the fig., a carry is generated from D3 bit position and propagates to the D4 position. This carry is called auxiliary carry.

S (Sign) Flag:

- Sign flag indicates whether the result of a mathematical operation is negative or positive.
- If the result is positive, then this flag will reset and if the result is negative this flag will be set.
- This bit, in fact, is a replica of the D7 bit.

P (Parity) Flag:

- Parity is the number of 1's in a number.
- If the number of 1's in a number is even then that number is known as even parity number.
- If the number of 1's in a number is odd then that number is known as an odd parity number.
- This flag indicates whether the current result is of even parity (set) or of odd parity (reset).

8085 Pin Configuration

1. Explain 8085 pin diagram

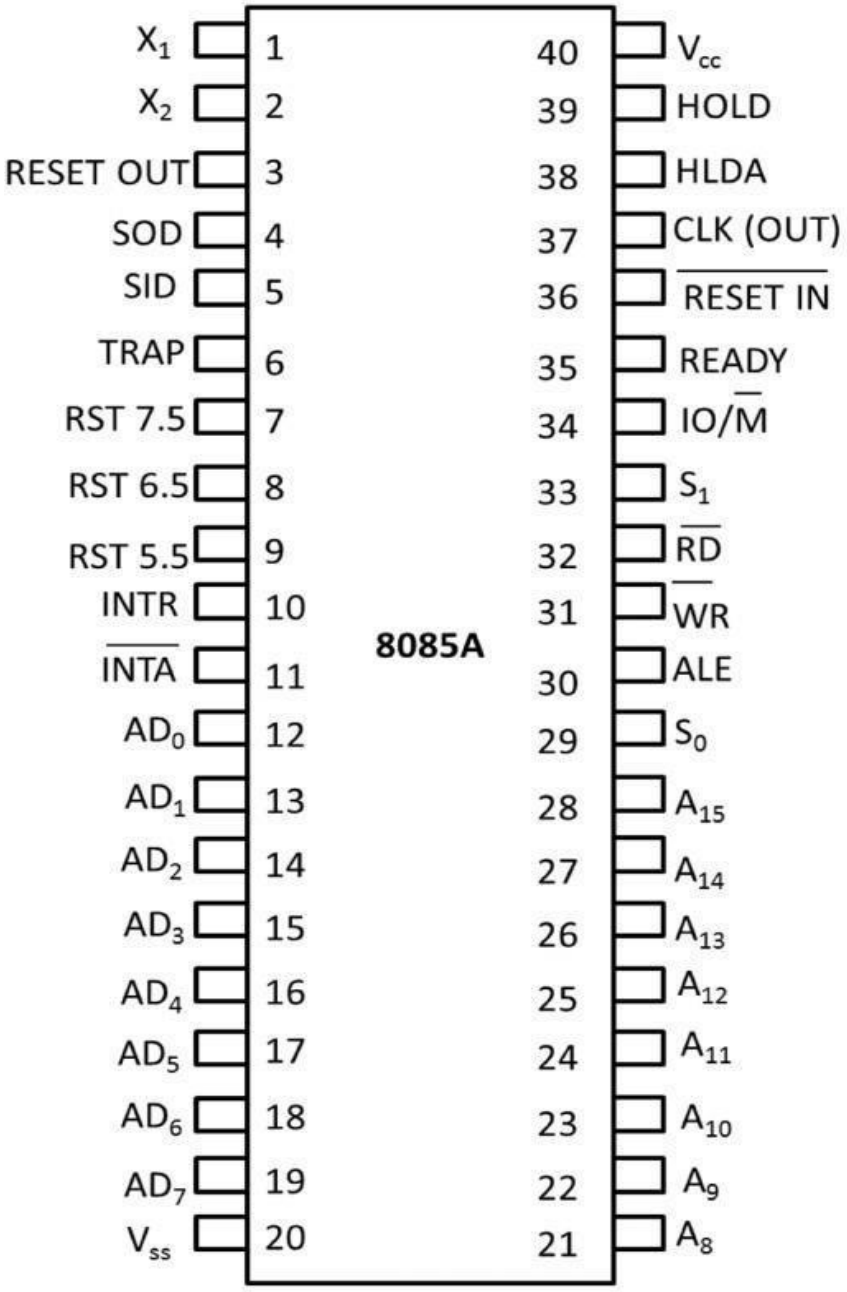


Figure: 8085 pin diagram.

- All signals can be classified into six groups:
  1. Address Bus
  2. Data Bus
  3. Control & Status Signals
  4. Power Supply & Frequency signals
  5. Externally initiated signals
  6. Serial I/O Ports

1) Address Bus (pin 12 to 28)

- 16 signal lines are used as address bus.
- However these lines are split into two segments: A15 - A8 and AD7 - AD0
- A15 - A8 are unidirectional and are used to carry high-order address of 16-bit address.
- AD7 - AD0 are used for dual purpose.

2) Data Bus/ Multiplexed Address (pin 12 to 19)

- Signal lines AD7-AD0 are bidirectional and serve dual purpose.
- They are used as low-order address bus as well as data bus.
- The low order address bus can be separate from these signals by using a latch.

3) Control & Status Signals

- To identify nature of operation
- Two Control Signals
  1) RD' (Read-pin 32)
     - ✓ This is a read control signal (active low)
     - ✓ This signal indicates that the selected I/O or Memory device is to be read & data are available on data bus.
  2) WR' (Write-pin 31)
     - ✓ This is a write control signal (active low)
     - ✓ This signal indicates that the selected I/O or Memory device is to be write.

- Three Status Signals 1) S1 (pin33)

2) S0 (pin 29)

  - ✓ S1 and S0 status signals can identify various operations, but they are rarely used in small systems.

S1	**S0**	**Mode**
0	0	HLT
0	1	WRITE
1	0	READ
1	1	OPCODE FETCH

3) IO/M' (pin 34)

  - ✓ This is a status signal used to differentiate I/O and memory operation
  - ✓ When it is high, it indicates an I/Ooperation
  - ✓ When it is low, it indicates a memoryoperation
  - ✓ This signal is combined with RD' and WR' to generate I/O & memory control signals

- To indicate beginning of operation
  - One Special Signal called ALE (Address Latch Enable-Pin 30)
  - This is positive going pulse generated every time the 8085 begins an operation (machinecycle)
  - It indicates that the bits on AD7-AD0 are address bits
  - This signal is used primarily to latch the low-address from multiplexed bus & generate a separate set of address lines A7-A0.

4) Power Supply & Frequency Signal

- V_{CC} Pin no. 40, +5V Supply
- V_{SS} Pin no.20, Ground Reference

- X1, X2 – Pin no.1 & 2, Crystal Oscillator is connected at these two pins. The frequency is internally divided by two;
  - Therefore, to operate a system at 3MHz, the crystal should have a frequency of 6MHz.
- CLK (OUT) – Clock output. Pin No.37: This signal can be used as the system clock for other devices.

5) Externally Initiated Signals including Interrupts

- INTR (Input) – Interrupt Request. It is used as general purpose interrupt

- INTA' (Output) – Interrupt Acknowledge. It is used to acknowledge an interrupt.

- RST7.5, RST6.5, RST5.5 (Input) – Restart Interrupts.
  - These are vector interrupts that transfer the program control to specific memory locations.
  - They have higher priorities than INTR interrupt.
  - Among these 3 interrupts, the priority order is RST7.5, RST6.5, RST5.5

- TRAP (Input) – This is a non maskable interrupt & has the highest priority.

- HOLD (Input) – This signal indicates that a peripheral such as DMA Controller is requesting the use of address & data buses

- HLDA (Output) – Hold Acknowledge. This signal acknowledges the HOLD request

- READY (Input) – This signal is used to delay the microprocessor read or write cycles until as low- responding peripheral is ready to send or accept data. When the signal goes low, the microprocessor waits for an integral no. of clock cycles until it goes high.

- RESET IN' (Input) When the signal on this pin goes low, the Program Counter is set to zero, the buses are tri-stated & microprocessor is reset.
- RESET OUT (Output) This signal indicates that microprocessor is being reset. The signal can be used to reset other devices.

6) Serial I/O Ports

- Two pins for serial transmission
  1) SID (Serial Input Data-pin 5)
  2) SOD (Serial Output Data-pin 4)
- In serial transmission, data bits are sent over a single line, one bit at a time.

Explain 8085 Programming Model

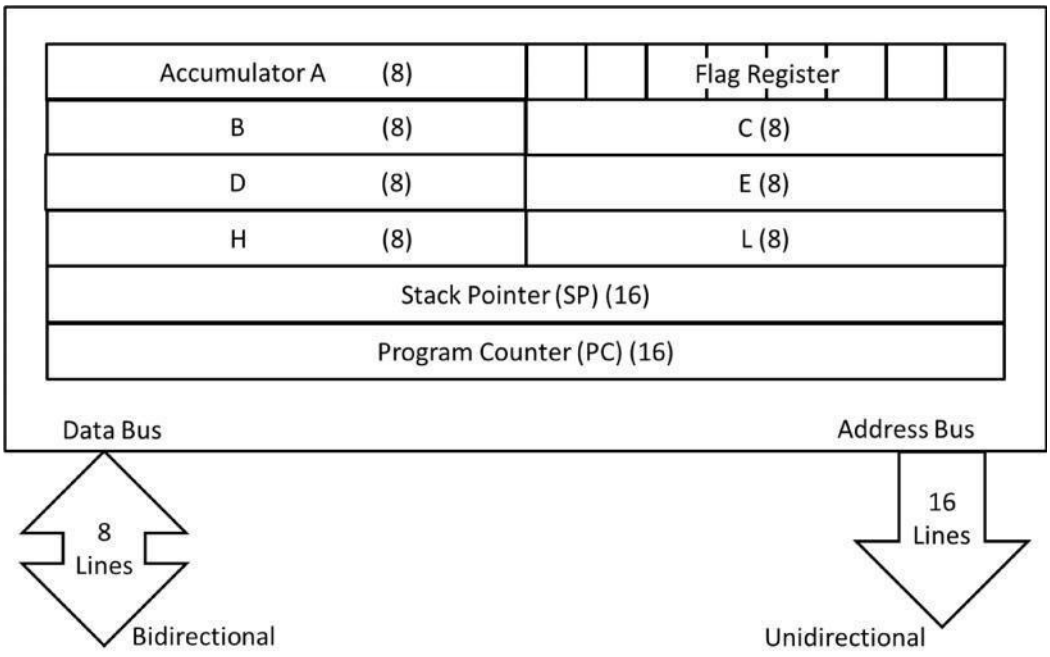


Figure: 8085 Programming Model.

Registers

- 6 general purpose registers to store 8-bit data B, C, D, E, H & L.
- Can be combined as register pairs – BC, DE, and HL to perform 16-bit operations.
- Used to store or copy data using data copy instructions.

Accumulator

- 8 - bit register, identified as A
- Part of ALU
- Used to store 8-bit data to perform arithmetic & logical operations.
- Result of operation is stored in it.

Flag Register

- ALU has 5 Flag Register that set/reset after an operation according to data conditions of the result in accumulator & other registers.
- Helpful in decision making process of Microprocessor
- Conditions are tested through software instructions
- For e.g.
- JC (Jump on Carry) is implemented to change the sequence of program when CY is set.

Program Counter

- 16-bit registers used to hold memory addresses.
- Size is 16-bits because memory addresses are of 16-bits.
- Microprocessor uses PC register to sequence the execution of instructions.
- Its function is to point to memory address from which next byte is to be fetched.
- When a byte is being fetched, PC is incremented by 1 to point to next memory location.

Stack Pointer

- Used as memory pointer
- Points to the memory location in R/W memory, called Stack.
- Beginning of stack is defined by loading a 16-bit address in the stack pointer.

Explain Bus Organization of 8085

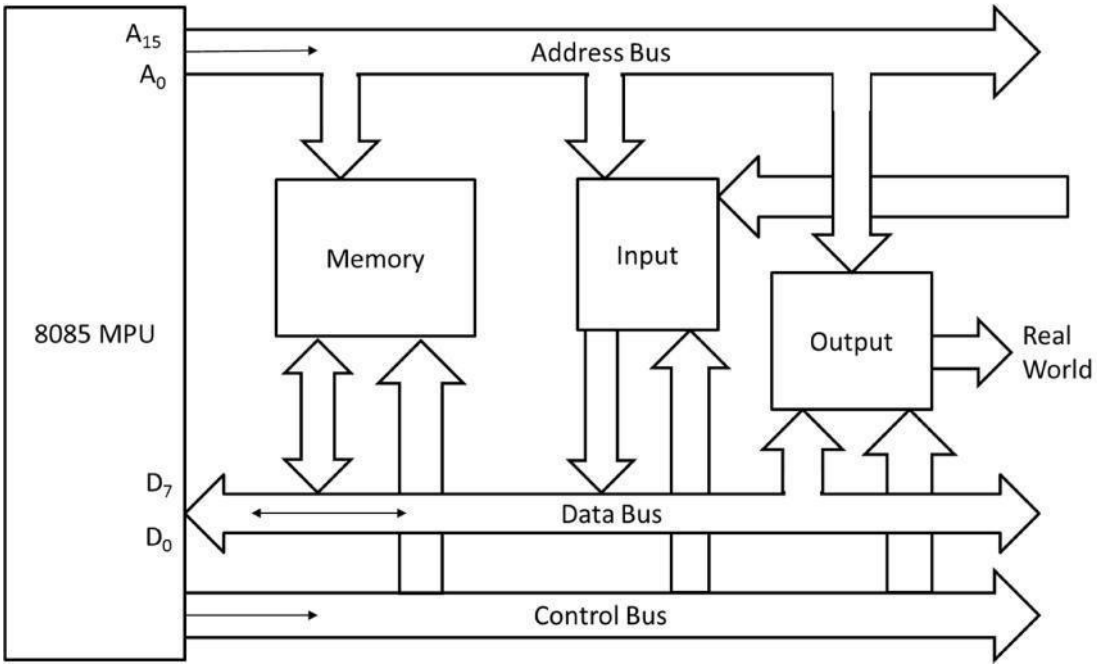


Figure: Bus Organization of 8085.

Address Bus

- Group of 16 lines generally identified as A0 to A15.
- It is unidirectional i.e. bits flow from microprocessor to peripheraldevices.
- 16 address lines are capable of addressing 65536 memorylocations.
- So, 8085 has 64K memory locations.

Data Bus

- Group of 8 lines identified as D0 to D7.
- They are bidirectional i.e. data flow in both directions between microprocessor, memory & peripheral.
- 8 data lines enable microprocessor to manipulate data ranging from 00H to FFH (28=256 numbers).
- Largest number appear on data bus is 1111 1111 =>(255)10.
- As Data bus is of 8-bit, 8085 is known as 8-bit Microprocessor.

Control Bus

- It comprises of various single lines that carry synchronization, timing & control signals

- These signals are used to identify a device type with which MPU intends to communicate.

Explain Demultiplexing AD0-AD7

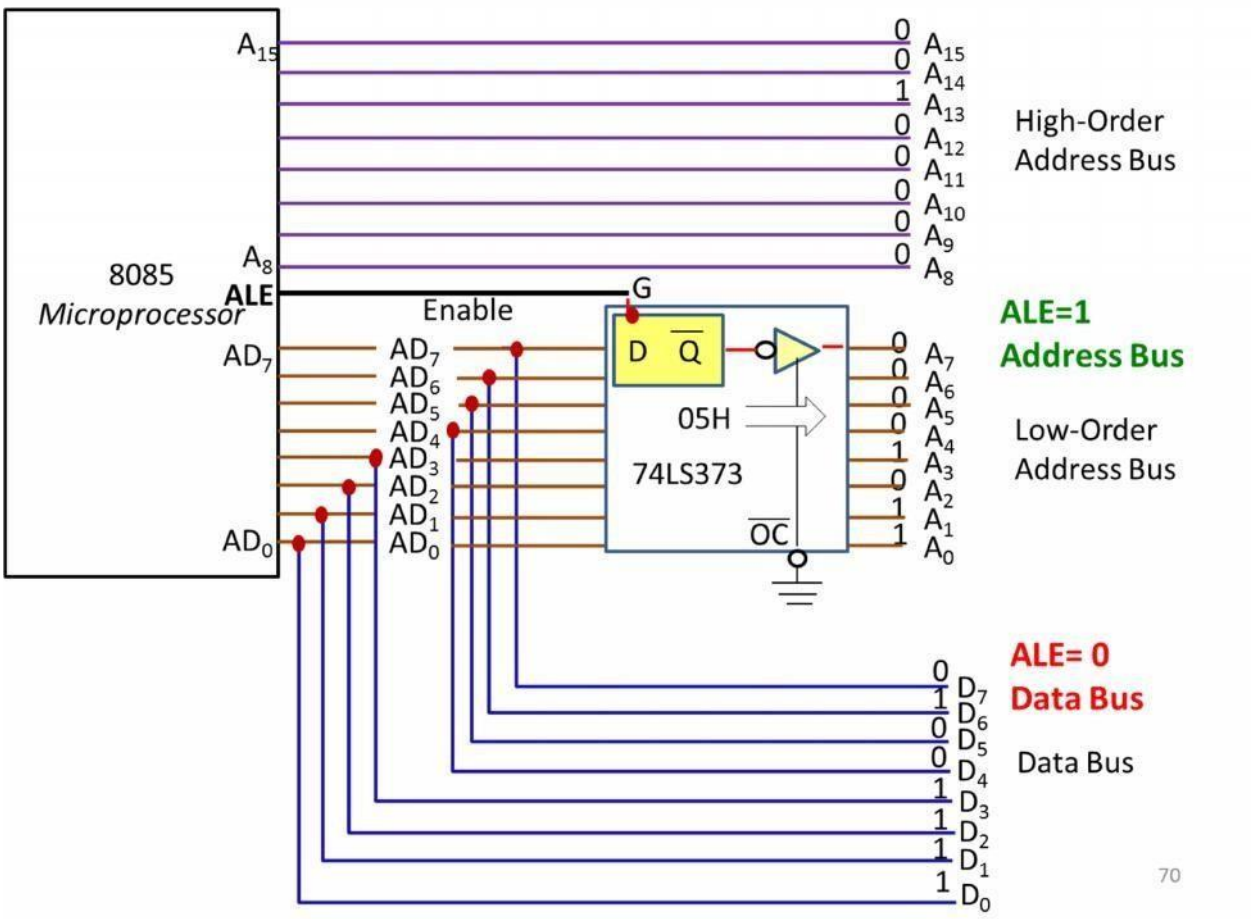


Figure: Demultiplexing AD0-AD7.

- The higher-order bus remains on the bus for three clock periods. However, the low-order address is lost after the first clock period.
- This address need to be latched and used for identifying the memory address. If the bus AD7- AD0 is used to identify the memory location (2005H), the address will change to 204FH after the first clock period.
- Figure shows a schematic that uses a latch and the ALE signal to demultiplex the bus.
- The bus AD7-AD0 is connected as the input to thelatch.
- The ALE signal is connected to the Enable pin of the latch, and the output control signal of the latch is grounded.
- Figure shows that the ALE goes high during T1. And during T1 address of lower-order address bus is store into the latch.

INSTRUCTION CYCLE

(1) Define following terms: Instruction, Machine Cycle, Opcode, Oprand & Instruction Cycle.

Instruction:

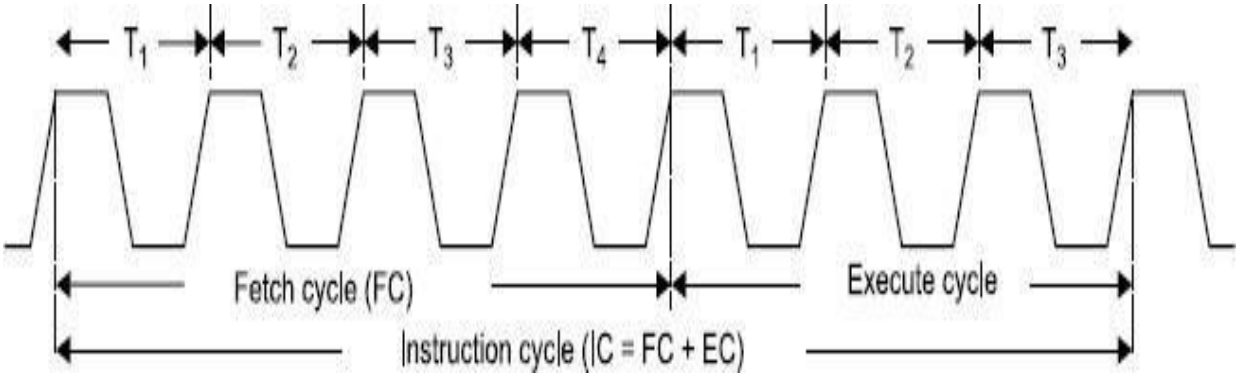
Instruction is the command given by the programmer to the Microprocessor to Perform the Specific task. For example, transfer a data, to do addition etc.

Machine Cycle:

Machine cycle is the time required to transfer data to or from memory or I/O devices. Each read or writes operation constitutes a machine cycle. The instructions of 8085 require 1–5 machine cycles containing 3–6 clocks. The 1st machine cycle of any instruction is always an Opcode fetching cycle in which the processor decides the nature of instruction. It is of at least 4-clocks. It may go up to 6-Clocks.

Instruction Cycle:

An instruction cycle is defined as the time required for fetching and executing an instruction. For executing any program, basically 3-steps are followed sequentially that is Fetch, Decode and Execute. The time taken by the μP in performing the fetch operations is called fetch cycle (Opcode fetch). The time taken by the μP in performing the execution operations is called execute cycle. Thus, sum of the fetch and execute cycle is called the instruction cycle as indicated in Fig



Instruction Cycle (IC) = Fetch cycle (FC) + Execute Cycle (EC)

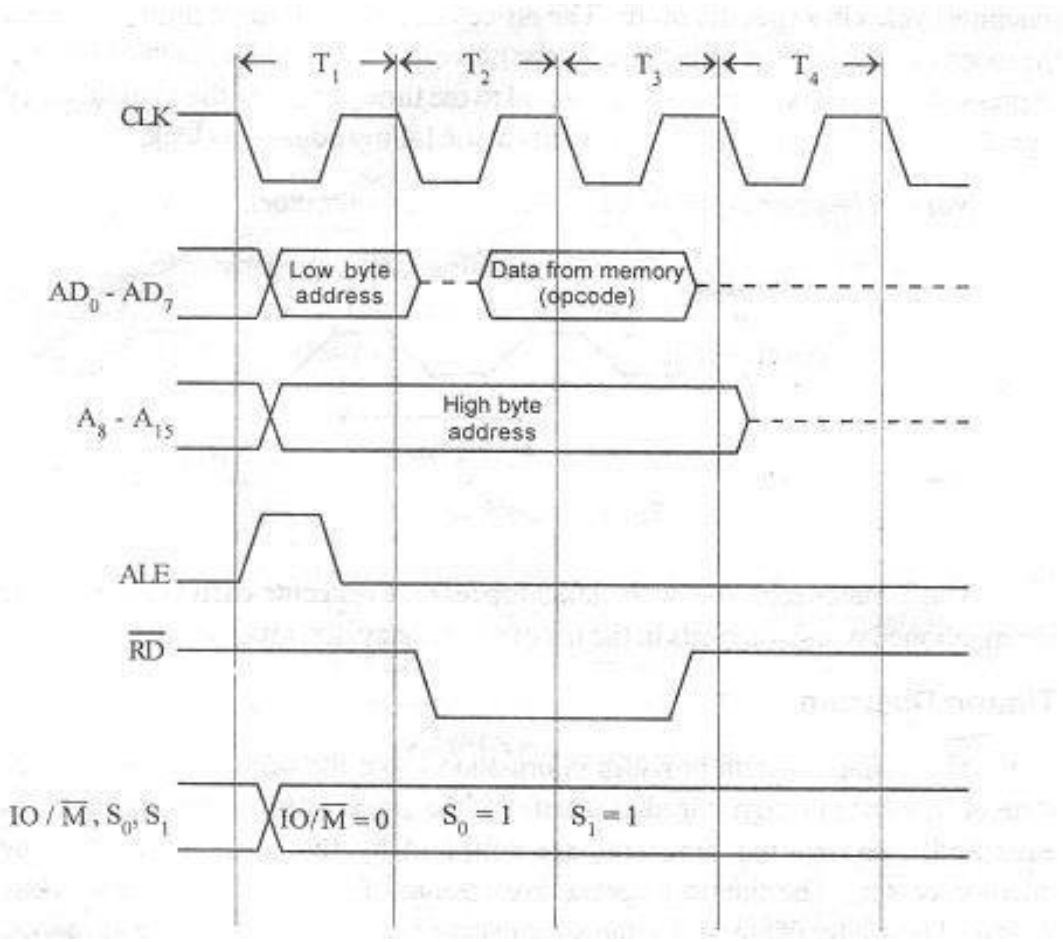
Opcode:

Operation Perform by the microprocessor is called Opcode.

Operand:

The Data on which Microprocessor perform operation is called Operand.

(2) Draw and explain the Timing Diagram for Opcode Fetch operation.



Timing diagram of Opcode fetch cycle is shown in figure.

Each instruction of the processor has one byte Opcode. The Opcode are stored in memory. So, the processor executes the Opcode fetch machine cycle to fetch (Read) the Opcode from memory. Hence, every instruction starts with Opcode fetch machine cycle.

The time taken by the processor to execute the Opcode fetch cycle is 4T or 6T.In this

time, the first, 3 T-states are used for fetching the Opcode from memory and the remaining T-states are used for internal operations by the processor.

T1 State: - In the T1 state, the microprocessor send the low byte address on AD0-AD7 lines and high byte address on A8 to A15 lines. ALE is send high to enable the address **latch. The other control signals are asserted as follows. IO/$\overline{M}$=0, S0=1, S1=1**

T2 State:-In the T2-state, the microprocessor send the $\overline{RD}$ to the memory. When $\overline{RD}$ is asserted to low the memory is enabled for placing the data on the data bus. The time allowed for memory to output the data is the time during which read remains low.

T3 State:-In third T3-state, the read signal is asserted high. On the rising edge of read signal the data is latched into microprocessor other control signals remains in the same state until the next machine cycle.

T4 State:-The T4-state is used by the processor for internal operations to decode the instruction and encode into various machine cycles, and also for completing the task specified by l byte instructions. During this cycle the address and data bus will be in high impedance state.

Explain memory read and Write operation with help of timing diagram.

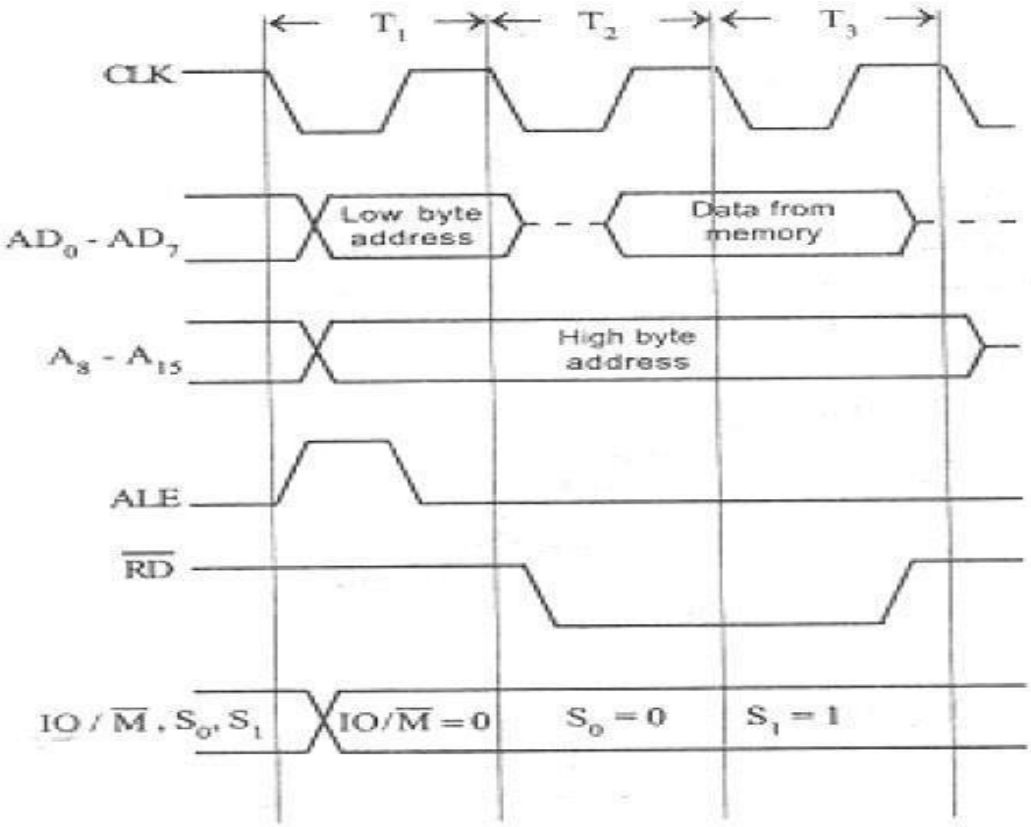
Memory read Operation:-

The memory read machine cycle is executed by the processor to read a data byte from memory. The processor takes 3T states to execute this cycle.

T1 State: - In the T1 state, the microprocessor send the low byte address on AD0-AD7 lines and high byte address on A8 to A15 lines. ALE is asserted high to enable the **address latch. The other control signals are asserted as follows. IO/$\overline{M}$=0, S0=0, S1=1**

T2 State:-In the T2-state, the microprocessor send the $\overline{RD}$ to the memory. When $\overline{RD}$ is asserted to low the memory is enabled for placing the data on the data bus. The time allowed for memory to output the data is the time during which read remains low.

T3 State:-In third T3-state, the read signal is asserted high. On the rising edge of read signal the data is latched into microprocessor other control signals remains in the same Timing diagram of Memory Read cycle is shown in figure.state until the next machine cycle.



Memory write Operation:-

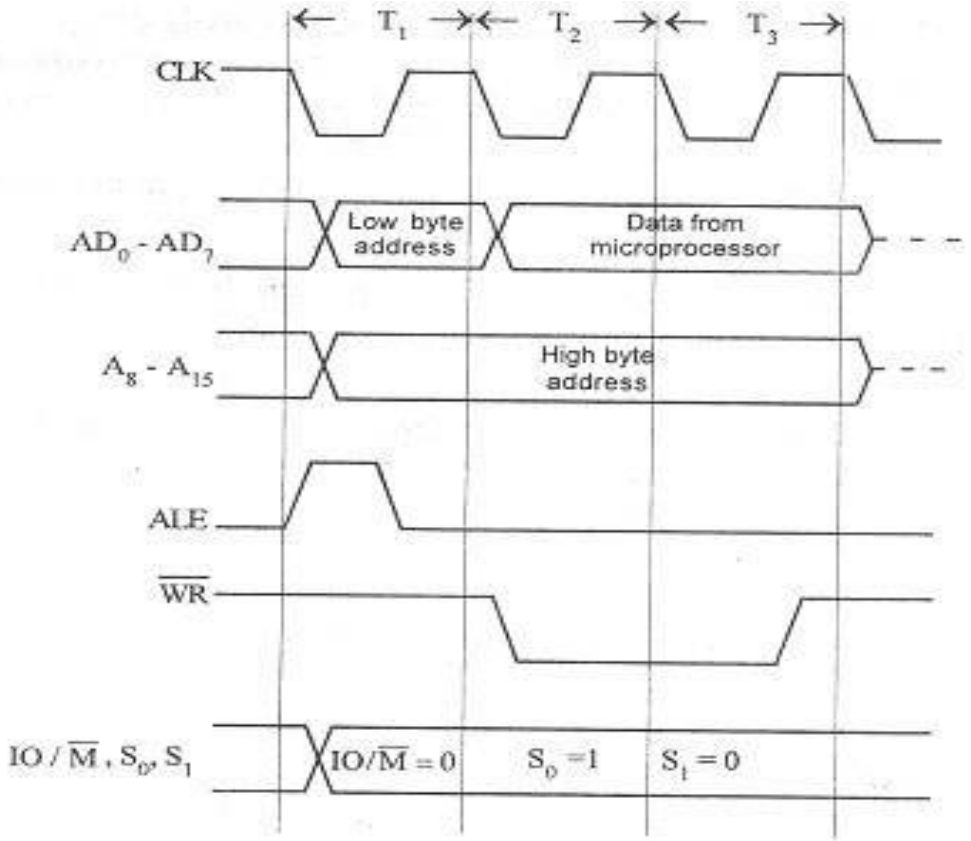
The memory write cycle is executed by processor to write a data byte in a memory location. The processor takes 3T states to execute this machine cycle.

T1 State: - In the T1 state, the microprocessor send the low byte address on AD0-AD7 lines and high byte address on A8 to A15 lines. ALE is asserted high to enable the address latch. The other control signals are asserted as follows. IO/$\overline{M}$=0, S0=1, S1=0

T2 State:-In the T2-state, the microprocessor send the $\overline{WR}$ to the memory. When $\overline{WR}$ is asserted to low the memory is enabled for placing the data on the data bus. The time allowed for memory to write the data is the time during which $\overline{WR}$ remains low.

T3 State:-In third T3-state, the write signal is asserted high. On the rising edge of write signal the data is write to memory other control signals remains in the same state until the next machine cycle.

The timing of various signals during memory write cycle is shown in fig below.



($\overline{RD}$ will be high ; READY is tied high either permanently or temporarily in the system.)

Memory write machine cycle of 8085

Explain I/O read and I/O Write operation with help of timing diagram.

I\O read Operation:-

I\O read machine cycle is executed by the processor to read a data byte from input **device. The processor takes 3T states to execute this cycle.**

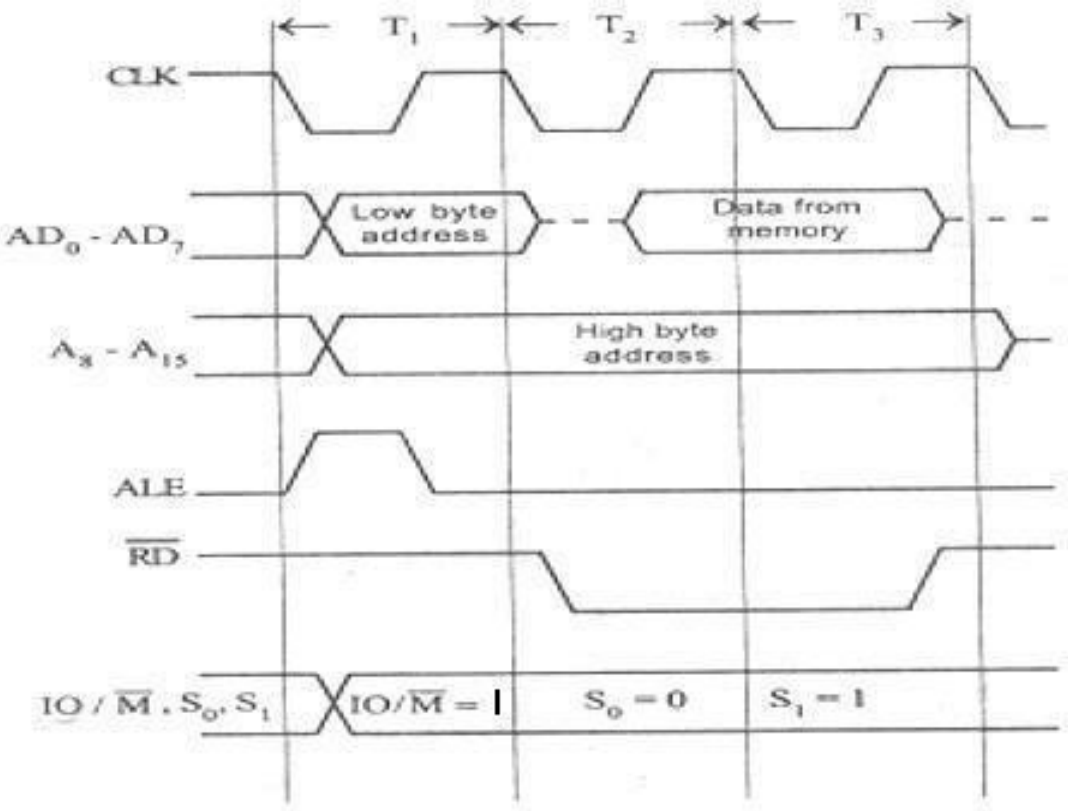
T1 State: - In the T1 state, the microprocessor send the low byte address on AD0-AD7 lines and high byte address on A8 to A15 lines. ALE is asserted high to enable the address latch. The other control signals are asserted as follows. IO/$\overline{M}$=1, S0=0, S1=1

T2 State:-In the T2-state, the microprocessor send the $\overline{RD}$ to the memory. When $\overline{RD}$ is asserted to

low the memory is enabled for placing the data on the data bus. The time allowed for memory to output the data is the time during which read remains low.

T3 State:-In third T3-state, the read signal is asserted high. On the rising edge of read signal the data is latched into microprocessor other control signals remains in the same state until the next machine cycle.

Timing diagram of I\O Read cycle is shown in figure.



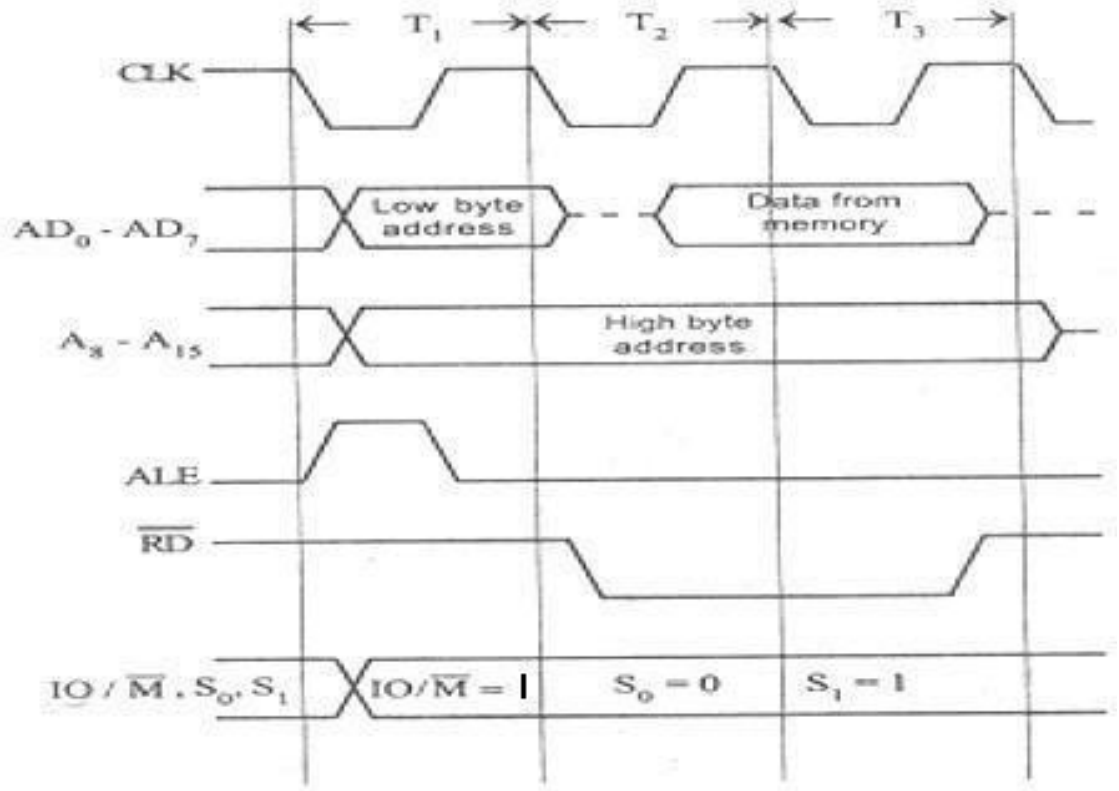
I\O write Operation:-

The I\O write cycle is executed by processor to write a data byte in a memory location. The processor takes 3T states to execute this machine cycle. The timing of various signals during memory write cycle is shown in fig below.

T1 State: - In the T1 state, the microprocessor send the low byte address on AD0-AD7 lines and high byte address on A8 to A15 lines. ALE is asserted high to enable the address latch. The other control signals are asserted as follows. IO/$\overline{M}$=1, S0=1, S1=0

T2 State:-In the T2-state, the microprocessor send the $\overline{WR}$ to the memory. When $\overline{WR}$ is asserted to low the memory is enabled for placing the data on the data bus. The time allowed for memory to write the data is the time during which $\overline{WR}$ remains low.

T3 State:-In third T3-state, the write signal is asserted high. On the rising edge of write signal the data is write to memory other control signals remains in the same state until the next machine cycle.



PRACTICE QUESTION

(1) Draw the timing diagram of MVI B, 40 instruction. OR Draw timing diagram for MVI A, 55H and explain. OR What is timing diagram? Draw timing diagram for MVI B, 30H instruction.

Timing diagram is the display of initiation of read/write and transfer of data operations under the control of 3-status signals IO /$\overline{M}$, S1, and S0.

MVI B, 40h is the two by instruction one is for opcode fetch and second is Memory Read Operation. Timing Diagram is shown in figure below.

UNIT - II

Microprocessor and Microcontroller – SPH1313

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. Direct addressing
III. Indirect addressing
IV. Memory direct 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.

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

Register Addressing:

Register direct addressing transfer a copy of a byte or word from source register to destination register.

Examples of Register addressing mode.

```
ADD B    ;Adds the value stored in B to the value in Accum.

ADC C    ;Contents of register C and carry flag are added to the

contents of Accumulator SUB E      ;Subtracts the value stored in

E to that in Accum.

SBB E    ;Contents of register E and carry flag are added to the contents

of Accumulator INR C          ;Increments the contents of register C by 1

INX B          ;Increments the contents of register pair BC by 1
```

Direct addressing mode

In this mode, the data is directly copied from the given address to the register.

```
LDA 2034H    ;Load the content at memory location 2034H into the
Accumulator

LHLD 2040H ;Load the content at 2040H into register L and contents at
the next location 2041H into register H

STA 3030H ;Load the content at memory location 3030H into the
Accumulator

SHLD 2040H ;Load the contents of the register L into memory location with
```

Indirect Addressing:

In this mode, the data is transferred from one register to another by using the address pointed by the register.

For example:

```
MOV A, M    ;move contents of the memory location whose address is
held by HL pair into the Accumulator
ADD M    ;Add contents of the memory location whose address is
held by HL pair to the contents of theAccumulator
```

```
INR M    ;Add contents of the memory location whose address is held by
HL pair is incremented by 1
CMP M    ;16 bit number from HL pair is picked up. Then, contents at
the address given by picked up number is compared with those of the
accumulator. 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
```

Implicit Addressing

This mode doesn’t require any operand; the data is specified by the opcode itself.

Example:

```
CMA ; Each bit of the 8 bit number stored in Accumulator is
complemented. Note that in this case, it is implied that the data to be
processed is in the accumulator and we don't need to specify it.
RLC ; the 8 bits of the number stored in the Accumulator are
manipulated in such a way that each bit is moved left by 1 place and the
most significant bit is moved to the least significant position.
```

INSTRUCTION SET OF 8085

- 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***.
- 8085 has **246** instructions.
- Each instruction is represented by an 8-bit binary value.
- These 8-bits of binary value is called ***Op-Code*** or ***Instruction Byte***.

Classification of Instruction Set

- Data Transfer Instruction
- Arithmetic Instructions
- Logical Instructions
- Branching Instructions
- Control Instructions

Data Transfer Instruction

- These instructions move data between registers, or between memory and registers.
- These instructions copy data from source to destination.
- While copying, the contents of source are not modified.

Opcode	Operand	Description
MOV	Rd, Rs Rd, M M, Rs	Copy from source to destination.

- 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

- MOV B, M
- MOV M, C

Data Transfer Instruction

Opcode	Operand	Description
MVI	Rd, Data M, Data	Move immediate 8-bit

- 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 H-Lregisters.

Example: MVI A, 57H

- MVI M, 57H

Data Transfer Instruction

Opcode	Operand	Description
LXI	Reg. pair, 16-bit data	Load register pair immediate

- This instruction loads 16-bit data in the register pair.

- **Example:** LXI H, 2034 H

Data Transfer Instruction

Opcode	Operand	Description
LDA	16-bit address	Load Accumulator

- 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

Data Transfer Instruction

Opcode	Operand	Description
LDAX	B/D Register Pair	Load accumulator indirect

- 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

Data Transfer Instruction

Opcode	Operand	Description
LHLD	16-bit address	Load H-L registers direct

- This instruction copies the contents of memory location pointed out by 16-bit address into register L.
- It copies the contents of next memory location into register H.
- **Example:** LHLD 2040 H

Data Transfer Instruction

Opcode	Operand	Description
STA	16-bit address	Store accumulator direct

- The contents of accumulator are copied into the memory location specified by the operand.
- **Example:** STA 2500 H

Data Transfer Instruction

Opcode	Operand	Description
STAX	Reg. pair	Store accumulator indirect

- The contents of accumulator are copied into the memory location specified by the contents of the register pair.
- **Example:** STAX B

Data Transfer Instruction

Opcode	Operand	Description
SHLD	16-bit address	Store H-L registers direct

- The contents of register L are stored intomemory location specified by the 16-bitaddress.
- The contents of register H are stored into the next memory location.
- **Example:** SHLD 2550 H

Data Transfer Instruction

Opcode	Operand	Description
XCHG	None	Exchange H-L with D-E

- The contents of register H are exchanged with the contents of register D.
- The contents of register L are exchanged with the contents of register E.
- **Example:** XCHG

Arithmetic Instructions

- These instructions perform the operations like:
- Addition
- Subtract
- Increment
- Decrement

Addition

- Any 8-bit number, or the contents of register, or the contents of memory location can be added to the contents of accumulator.
- The result (sum) is stored in the accumulator.
- No two other 8-bit registers can be added directly.
- **Example:** The contents of register B cannot be added directly to the contents of register C.

Subtract

- Any 8-bit number, or the contents of register, or the contents of memory location can be subtracted from the contents of accumulator.
- The result is stored in the accumulator.
- Subtraction is performed in 2's complement form.
- If the result is negative, it is stored in 2's complement form.
- No two other 8-bit registers can be subtracted directly.

Increment/Decrement

- The 8-bit contents of a register or a memory location can be incremented or decremented by 1.
- The 16-bit contents of a register pair can be incremented or decremented by 1.
- Increment or decrement can be performed on any register or a memory location.

Arithmetic Instructions

Opcode	Operand	Description
ADD	R, M	Add register or memory to accumulator

- The contents of register or memory are added to the contents of accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.
- All flags are modified to reflect the result of the addition.
- **Example:** ADD B or ADD M

Arithmetic Instructions

Opcode	Operand	Description
ADC	R M	Add register or memory to accumulator with carry

- The contents of register or memory and Carry Flag (CY) are added to the contents of accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.
- All flags are modified to reflect the result of the addition.
- **Example:** ADC B or ADC M

Arithmetic Instructions

Opcode	Operand	Description
ADI	8-bit data	Add immediate to accumulator

The 8-bit **Arithmetic Instructions**

- data is added to the contents of accumulator.
- The result is stored inaccumulator.
- All flagsare modified to reflect the result of the addition.
- **Example:** ADI 45 H

Arithmetic Instructions

Opcode	Operand	Description
ACI	8-bit data	Add immediate to accumulator with carry

- The 8-bit data and the Carry Flag (CY) are added to the contents of accumulator.

- The result is stored in accumulator.
- All flags are modified to reflect the result of the addition.
- **Example:** ACI 45 H

Arithmetic Instructions

Opcode	Operand	Description
DAD	Reg. pair	Add register pair to H-L pair

- The 16-bit contents of the register pair are added to the contents of H-L pair.
- The result is stored in H-L pair.
- If the result is larger than 16 bits, then CY is set.
- No other flags are changed.
- **Example:** DAD B

Arithmetic Instructions

Opcode	Operand	Description
SUB	R M	Subtract register or memory from accumulator

- The contents of the register or memory location are subtracted from the contents of the accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.
- All flags are modified to reflect the result of subtraction.
- **Example:** SUB B or SUB M

Arithmetic Instructions

Opcode	Operand	Description
SBB	R M	Subtract register or memory from accumulator with borrow

- The contents of the register or memory location and Borrow Flag (i.e. CY) are subtracted from the contents of the accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.
- All flags are modified to reflect the result of subtraction.
- **Example:** SBB B or SBB M

Arithmetic Instructions

Opcode	Operand	Description
SUI	8-bit data	Subtract immediate from accumulator

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

Arithmetic Instructions

Opcode	Operand	Description
SBI	8-bit data	Subtract immediate from accumulator with borrow

- The 8-bit data and the Borrow Flag (i.e. CY) issubtracted from the contents of the accumulator.
- The result is stored in accumulator.
- All flags are modified to reflect the result ofsubtraction.
- **Example:** SBI 45 H

Arithmetic Instructions

Opcode	Operand	Description
INR	R M	Increment register or memory by 1

- The contents of register or memory location are incremented by 1.
- The result is stored in the same place.
- If the operand is a memory location, its address is specifiedby the contents of H-L pair.
- **Example:** INR B or INR M

Arithmetic Instructions

Opcode	Operand	Description
INX	R	Increment register pair by 1

- The contents of register pairare incremented by 1.
- The result is stored in the sameplace.
- **Example:** INX H

Arithmetic Instructions

Opcode	Operand	Description
DCR	R M	Decrement register or memory by 1

- The contents of register or memory location are decremented by 1.
- The result is stored in the same place.
- If the operand is a memory location, its address is specifiedby the contents of H-L pair.
- **Example:** DCR B or DCR M

Arithmetic Instructions

Opcode	Operand	Description
DCX	R	Decrement register pair by 1

- The contents of register pair are decremented by 1.
- The result is stored in the sameplace.
- **Example:** DCX H

LOGICAL INSTRUCTIONS

These instructions perform logical operations on data stored in registers, memory and status flags.

The logical operations are:

- AND
- OR
- XOR
- Rotate
- Compare
- Complement

AND, OR, XOR

Any 8-bit data, or the contents of register, or memory location can logically have

- AND operation
- OR operation
- XOR operation with the contents of accumulator.
- The result is stored in accumulator.

ROTATE

- Each bit in the accumulator can be shifted either left or right to the next position.

COMPARE

Any 8-bit data, or the contents of register, or memory location can be compares for:

- Equality
- Greater Than
- Less Than with the contents of accumulator.
- The result is reflected in status flags.

COMPLEMENT

- The contents of accumulator can be complemented.
- Each 0 is replaced by 1 and each 1 is replaced by0.

LOGICAL INSTRUCTION

Opcode	Operand	Description
CMP	R M	Compare register or memory with accumulator

- The contents of the operand (register or memory) are compared with the contents of theaccumulator.
- Both contents are preserved.
- The result of the comparison is shown bysetting the flags of the PSW as follows:

Opcode	Operand	Description
CMP	R, M	Compare register or memory with accumulator

- if (A) < (reg/mem): carry flag isset
- if (A) = (reg/mem): zero flag isset
- if (A) > (reg/mem): carry and zero flags arereset.
- **Example:** CMP B or CMP M

LOGICAL INSTRUCTION

Opcode	Operand	Description
CPI	8-bit data	Compare immediate with accumulator

- The 8-bit data is compared with the contents of accumulator.
- The values being compared remainunchanged.
- The result of the comparison is shown bysetting the flags of the PSW as follows:

Opcode	Operand	Description
CPI	8-bit data	Compare immediate with accumulator

- if (A) < data: carry flag isset
- if (A) = data: zero flag isset
- if (A) > data: carry and zero flags arereset
- **Example:** CPI 89H

LOGICAL INSTRUCTION

Opcode	Operand	Description
ANA	R M	Logical AND register or memory with accumulator

- The contents of the accumulator are logically ANDed with the contents of register or memory.
- The result is placed in the accumulator.
- If the operand is a memory location, its address is specified by thecontents of H-L pair.
- S, Z, P are modified to reflect the result of the operation.
- CY is reset and AC is set.
- **Example:** ANA B or ANA M.

LOGICAL INSTRUCTION

Opcode	Operand	Description
ANI	8-bit data	Logical AND immediate with accumulator

- The contents of the accumulator are logically ANDed withthe 8-bit data.
- The result is placed in the accumulator.
- S, Z, P are modified to reflect the result.
- CY is reset, AC is set.
- **Example:** ANI 86H.

LOGICAL INSTRUCTION

Opcode	Operand	Description
XRA	R M	Exclusive OR register or memory with accumulator

- The contents of the accumulator are XORed with the contents of the register or memory.
- The result is placed in the accumulator.
- If the operand is a memory location, its address is specified by the contents of H-L pair.
- S, Z, P are modified to reflect the result of the operation.
- CY and AC are reset.
- **Example:** XRA B or XRA M.

LOGICAL INSTRUCTION

Opcode	Operand	Description
ORA	R M	Logical OR register or memory with accumulator

- The contents of the accumulator are logically OR ed with the contents of the register or memory.
- The result is placed in the accumulator.
- If the operand is a memory location, its address is specified by the contents of H-L pair.
- S, Z, P are modified to reflect the result.
- CY and AC are reset.
- **Example:** ORA B or ORA M.

LOGICAL INSTRUCTION

Opcode	Operand	Description
ORI	8-bit data	Logical OR immediate with accumulator

- The contents of the accumulator are logically Red with the8- bit data.
- The result is placed in the accumulator.
- S, Z, P are modified to reflect the result.
- CY and AC are reset.
- **Example:** ORI 86H.

LOGICAL INSTRUCTION

Opcode	Operand	Description
XRA	R M	Logical XOR register or memory with accumulator

- The contents of the accumulator are XORed with the contents of the register or memory.
- The result is placed in the accumulator.
- If the operand is a memory location, its address is specified bythe contents of H-L pair.
- S, Z, P are modified to reflect the result of the operation.
- CY and AC are reset.
- **Example:** XRA B or XRA M.

LOGICAL INSTRUCTION

Opcode	Operand	Description
XRI	8-bit data	XOR immediate with accumulator

- The contents of the accumulator are XORed with the 8-bit data.
- The result is placed in theaccumulator.
- S, Z, P are modified to reflect theresult.
- CY and AC arereset.
- **Example:** XRI 86H.

LOGICAL INSTRUCTION

Opcode	Operand	Description
RLC	None	Rotate accumulator left

- 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 theCarry flag.
- CY is modified according to bit D7.
- S, Z, P,AC are not affected.
- **Example:** RLC.

LOGICAL INSTRUCTION

Opcode	Operand	Description
RRC	None	Rotate accumulator right

- 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 theCarry flag.
- CY is modified according to bit D0.
- S, Z, P,AC are not affected.
- **Example:** RRC.

LOGICAL INSTRUCTION

Opcode	Operand	Description
RAL	None	Rotate accumulator left through carry

- 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.

LOGICAL INSTRUCTION

Opcode	Operand	Description
RAR	None	Rotate accumulator right through carry

- 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.

LOGICAL INSTRUCTION

Opcode	Operand	Description
CMA	None	Complement accumulator

- The contents of the accumulator are complemented.
- No flags areaffected.
- **Example:** CMA.

LOGICAL INSTRUCTION

Opcode	Operand	Description
CMC	None	Complement carry

-

- The Carry flag iscomplemented.
- No other flags areaffected.
- **Example:** CMC.

LOGICAL INSTRUCTION

Opcode	Operand	Description
STC	None	Set carry

- The Carry flag is set to1.
- No other flags areaffected.
- **Example:** STC.

BRANCH INSTRUCTIONS

The branching instruction alter the normal sequential flow.

These instructions alter either unconditionally or conditionally

BRANCH INSTRUCTIONS

Opcode	Operand	Description
JMP	16-bit address	Jump unconditionally

- The program sequence is transferred to thememory location specified by the 16-bit address given in the operand.
- **Example:** JMP 2034 H.

BRANCH INSTRUCTIONS

Opcode	Operand	Description
Jx	16-bit address	Jump conditionally

- The program sequence is transferred to thememory location specified by the 16-bit address given in the operand based on the specified flag of thePSW.
- **Example:** JZ 2034 H.
- The program sequence is transferred to thememory location specified by the 16-bit address given in the operand based on the specified flag of thePSW.
- 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 H.

JUMP CONDITIONALLY

Opcode	Description	Status Flags
JC	Jump if Carry	CY = 1
JNC	Jump if No Carry	CY = 0
JP	Jump if Positive	S = 0
JM	Jump if Minus	S = 1
JZ	Jump if Zero	Z = 1
JNZ	Jump if No Zero	Z = 0
JPE	Jump if Parity Even	P = 1
JPO	Jump if Parity Odd	P = 0

JUMP UNCONDITIONALLY

Opcode	Operand	Description
CALL	16-bit address	Call unconditionally

- The program sequence is transferred to thememory 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 pushedonto the stack.
- **Example:** CALL 2034 H.

RETURN UNCONDITIONALLY

Opcode	Operand	Description
RET	None	Return unconditionally

- The program sequence is transferred fromthe subroutine to the callingprogram.
- The two bytes from the top of the stack are copied into the program counter, and program execution beginsat the new address.
- **Example:** RET.

RETURN CONDITIONALLY

Opcode	Operand	Description
Rx	None	Call conditionally

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

Opcode	Description	Status Flags
RC	Return if Carry	CY = 1

RNC	Return if No Carry	CY = 0
RP	Return if Positive	S = 0
RM	Return if Minus	S = 1
RZ	Return if Zero	Z = 1
RNZ	Return if No Zero	Z = 0
RPE	Return if Parity Even	P = 1
RPO	Return if Parity Odd	P = 0

Opcode	**Operand**	**Description**
RST	0 – 7	Restart (Software Interrupts)

- The RST instruction jumps the control to one of eight memory locations depending upon thenumber.
- These are used as software instructions in a program to transfer program execution to one of the eight locations.
- **Example:** RST 3.

Opcode	**Operand**	**Description**
RST	0 – 7	Restart (Software Interrupts)

- The RST instruction jumps the control to one of eight memory locations depending upon thenumber.
- These are used as software instructions in a program to transfer program execution to one of the eight locations.
- **Example:** RST 3.

RESRART ADDRESSES

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

CONTROL INSTRUCTIONS

The control instructions control the operation of microprocessor.

Opcode	Operand	Description
NOP	None	No operation

- No operation isperformed.
- The instruction is fetched and decoded butno operation is executed.
- **Example:** NOP

CONTROL INSTRUCTIONS

Opcode	Operand	Description
HLT	None	Halt

- The CPU finishes executing thecurrent instruction and halts any further execution.
- An interrupt or reset is necessary to exit from the halt state.
- **Example:** HLT

CONTROL INSTRUCTIONS

Opcode	Operand	Description
DI	None	Disable interrupt

- The interrupt enable flip-flop is reset and all the interrupts except the TRAP aredisabled.
- No flags areaffected.
- **Example:** DI

CONTROL INSTRUCTIONS

Opcode	Operand	Description
EI	None	Enable interrupt

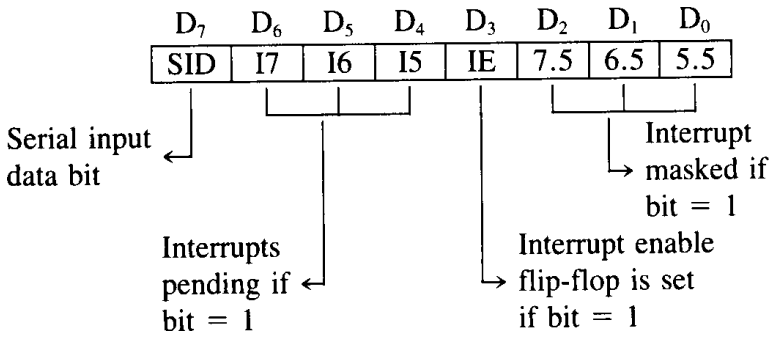
- The interrupt enable flip-flop is set and all interrupts are enabled.
- No flags areaffected.
- This instruction is necessary to re-enable the interrupts (except TRAP).
- **Example:** EI

CONTROL INSTRUCTIONS

Opcode	Operand	Description
RIM	None	Read Interrupt Mask

- 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

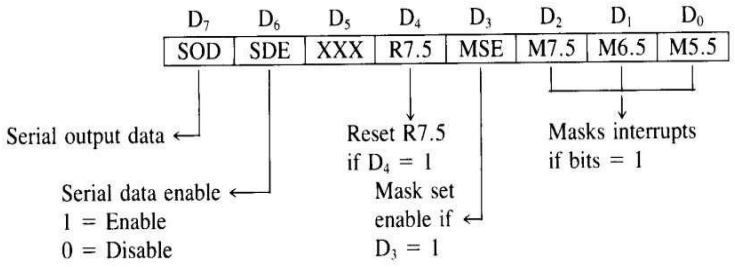
RIM Instruction



SIM Instruction

Opcode	Operand	Description
SIM	None	Set Interrupt Mask

- 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



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 triggered	Interrupt vector address	Maskable or non- maskable	Edge or level 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.

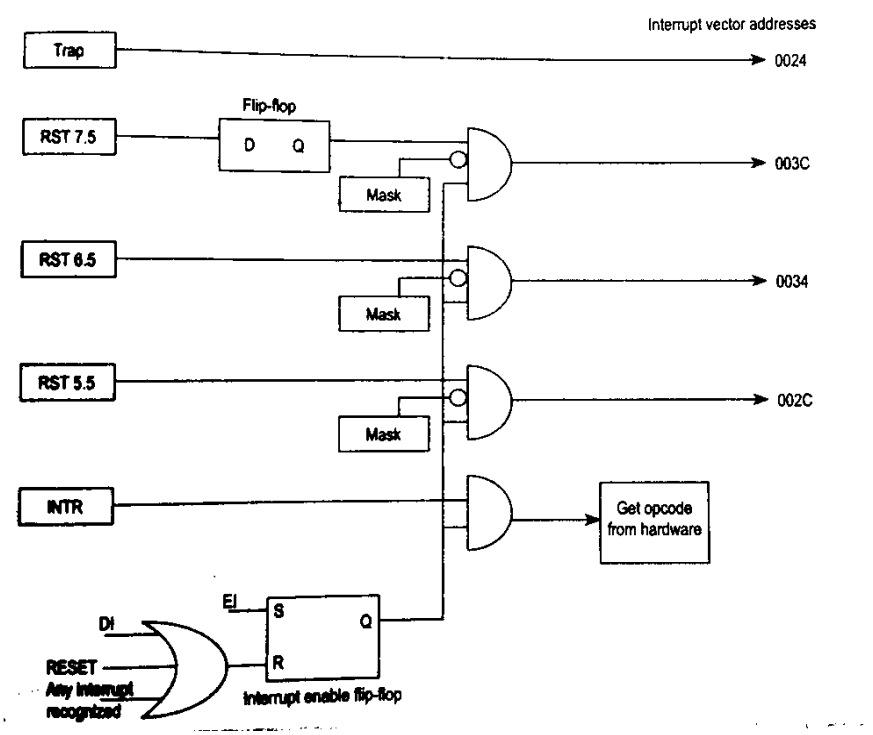


Fig. 13 Interrupt structure of 8085

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 Bit 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.

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

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.

ASSEMBLY LANGUAGE PROGRAMMING

1. Write a program to transfer a block of data from one location to the other.

```
5000 Start   LXI   B, 4A01
             LXI   H, 5101
             MVI   D,05
  Loop       MOV   A, M
             STAX  B
              INX  H
             INX   B
             DCR   D
             JNZ   Loop
             HLT
```

2. Write an assembly language program to add two 8 bit umbers.

1) Start the program by loading the first data into Accumulator.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.

4) Add the two register contents.
5) Check for carry.
6) Store the value of sum and carry in memory location.
7) Terminate the program.

MVI	C, 00	Initialize C register to 00
LDA	4150	Load the value to Accumulator.
MVI	C, 00	Initialize C register to 00
LDA	4150	Load the value to Accumulator.
MOV	B, A	Move the content of Accumulator to B register.
LDA	4151	Load the value to Accumulator.
ADD	B	Add the value of register B to A
JNC	LOOP	Jump on no carry.
INR	C	Increment value of register C
LOOP : STA	4152	Store the value of Accumulator (SUM).
MOV	A, C	Move content of register C to Acc.
STA	4153	Store the value of Accumulator (CARRY)
HLT		Halt the program.

3. Write an assembly language program to subtract two 8 bit numbers.

- Start the program by loading the first data into Accumulator.
- Move the data to a register (B register).
- Get the second data and load into Accumulator.
- Subtract the two register contents.
- Check for carry.
- If carry is present take 2's complement of Accumulator.
- Store the value of borrow in memory location.
- Store the difference value (present in Accumulator) to a memory
- location and terminate the program.

		MVI	C, 00	Initialize C to 00
		LDA	4150	Load the value to Acc.
		MOV	B, A	Move the content of Acc to B register.
		LDA	4151	Load the value to Acc.
4.		SUB	B	
5.		JNC	LOOP	Jump on no carry.
6.		CMA		Complement Accumulator contents.
7.		INR	A	Increment value in Accumulator.
		INR	C	Increment value in register C
	LOOP:	STA	4152	Store the value of A-reg to memory address.
		MOV	A, C	Move contents of register C to Accumulator.
4.		STA	4153	Store the value of Accumulator memory address.
5.		HLT		Terminate the program.
6.				

3. Subtraction two 8-bit BCD number using 8085

1 Perform subtraction by tens complement method
2 Take nine's complement of second no.(99-no)
3 Add one to nine's complement [(99-no) +1] to get 10's complement
4 Add with first no.
5 Convert to BCD using DAA instr.
6 Store in memory location.

LDA	2050 H	Load the first number to accumulator from Memory
MOV B	A	Store the number in B reg.
LDA	2051H	Load the second number to accumulator from memory
MOV C	A	Store the number in C reg.

MVI A	99H	Load acc. With 99H
SUB	C	Subtract second no from C reg.
ADD	B	Add the content with B reg.
DAA		Convert to BCD using DAA instr.
STA	5052	Store in memory location.
HLT		Halt the program.

4. Write an assembly language program to add two 16 bit numbers.

2050	
2051	

2060	
2061	

1. Clear the content in accumulator
2. Set the no. of bytes to be added in C reg.
3. Point to the first no.memory location by loading the address in HL reg. pair
4. Point to the second no.memory location by loading the address in DE reg. pair.
5. Add the first byte and store in first memory location
6. Decrement the counter reg. ; check for zero
6. Until zero continue adding
7. HLT

XRA	A	Clear the acc.
MVI C	02H	Add 02H immediate data in C reg.
LXI H	2050H	Load HL reg. pair with first memory location address
LXI D	2060H	Load DE reg. pair with second memory location address
HERE	LDAX D	load the content from memory whose address is

		in DE reg. pair
ADC	M	Add with carry with the content in acc.
MOV	M,A	Copy the content from acc. to memory location whose address is in HL reg.pair
INX	H	Increment the content in HL reg.pair
INX	D	; Decrement the content in DE reg.pair DCR C; Increment the content in C reg.
JNZ	HERE	: Continue the process from HERE; until zero
HLT		Halt the program.

5. Write an assembly language program to subtract two 16 bit numbers.

1. Load the first no.from memory location to accumulator
2. Store it in B reg.
3. Load the second no.from memory
4. Subtract with first no.
5. Check for carry
6. If carry is produced; increment C reg.
7. Store the LSB and MSB to memory location.

LDA	2050 H	Load the first no.from memory location to accumulator
MOV B	A	Move the content from Acc. to B reg
LDA	2051H	Load the second no.from memory location to accumulator
MVI C	OOH	Clear C reg
SUB	B	Subtract the content from acc. with B reg
JNC	GOTO	Continue until Carry
INR	C	increment the content in C reg.
GOTO:	STA 2052H	Store the content in acc. to memory (LSB)

MOV A	C	Copy the content from C.reg. to acc.(MSB
STA	2053H	Store the content from acc. to memory location(MSB)
HLT		End program

6. Write an assembly language program to subtract two 8 bit BCD numbers.

```
LDA 2050 H
 MOV B,A
LDA 2051H
MOV C,A
MVI A,99H
 SUB C
INR A
ADD B
DAA
STA 2052H
 HLT
```

7. Write an assembly language program to program to multiply two 8 – bit numbers.

8 – bit multiplication:

1) Start the program by loading the multiplicand into Accumulator.
2) Move the data to a register (B register).
3) Get the multiplier loaded into Accumulator.
4) Clear acc. and set reg. to store the catty (D register).

5) Add the multiplicand register contents and check for carry.
6) Decrement the . multiplier register contents and check for zero
7) Store the value of product and carry in memory location.
8) Terminate the program.

Start	LDA	2050H	Load the multiplicand from memory to accumulator
	MOV B	A	Copy the multiplicand to B reg.
	LDA	2051H	Load the multiplier from memory to accumulator
	MOV C	A	Copy the multiplier to C reg.,
	XRA	A	Clear Accumulator
	MOV D	A	Clear D reg.
HERE:	ADD	C	Add the content in C reg to accumulator
	JNC	GOTO	Check for carry, if no carry then decrement B reg
	INR	D	If carry exist, then increment D reg.
GOTO	DCR	B	decrement the content in B reg
	JNZ	HERE	Check for zero; Jump to HERE location until zero flag is set.
	STA	5052H	If zero flag is set, store the content in memory
	MOV	A, D	Move content of register D to Acc.
	STA	5053H	Store the value of Accumulator (CARRY)
	HLT		

8. Write an assembly language program to program to multiply two 8 – bit BCD numbers.

8 -Bit BCD Multiplication:

Subtraction using addition method; since DAA can be used only with

1. addition operator
2. Subtract by 9's complement menthod (i.e add 99 with the subtractant and then add with the subtractor)

	LDA	2050H	Load the subtract ant from memory location to accumulator
	MOV B	A	Copy the multiplicand to B reg.
	LDA	2051H	Load the subtractor from memory location to accumulator
	MOV C	A	Copy the multiplier to C reg.,
	XRA	A	Clear Accumulator
HERE	ADD	B	Add the content from B reg. to acc.
	DAA		Adjust to correct BCD no.
	MOV D	A	Store the content from acc. to D reg.
	MOV A	C	Copy the subtractor from C.reg to acc
	ADI	99H	add 99H
	DAA		Adjust to correct BCD no.
	MOV C	A	; Copy the content from acc. to C reg
	MOV	A, D	Move content of register D to Acc.
	JNZ	HERE	continue from step HERE; until zero occurs
	STA	5052H	store the content from acc. to

		memory loaction
	HLT	end the program.

9. Write an assembly language program to multiply two 16 – bit numbers.

A2050	HEX No.
2051	
2052	
2053	

1. Clear the memory location to load the numbers to be added
2. Load the HL reg. and BC reg. pair with the 16 bit no. to be added
3. Add reg. pair (BC with HL)
4. Check for carry; Then, increment the content in 2052 memory location If carry is generated to save the carry in memory;
5. Decrement BC reg.pair; Check for zero;
6. Else, to add the next byte; increment the SP to point to next memory location
7. To load the PUSH the content in HL to stack and then load the carry

	LXI H	0000H	; Clear HL reg.
	SHLD	2050H	Clear memory location 2050H by storing the content in HL to memory .
	LXI B	1234H	Load HL reg.pair with first no.
	LXI D	1001H	Load BC reg.pair with second no.
LOOP2	:DAD	B	; ADD the content of BC reg. pair with HL reg. pair

	JNC	LOOP1	Check for carry;l If carry is produced jump to LOOP 1 PUSH H; Else, Push the content in HL reg.pair to stack
			.
	LHLD	2052H	Load HL reg.pair with next byte no.
	INX	H	Increment the content in HL reg. pair by one
	SHLD	2052H	Store the carry to memory
	POP	H	now retrieve the content from stack to HL reg.pair
LOOP 1	:DCX	D	Decrement D.Reg.
	MOV A	E	Copy the content from E reg. to acc
			ORA E; OR the content of acc. with E reg.
	JNZ	LOOP 2	If NO zero is produced jump to LOOP 2
	SHLD	2050H	store the result from HL reg. pair to memory location 2050H
	HLT		end the program.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF SCIENCE AND HUMANITIES

DEPARTMENT OF PHYSICS

UNIT - III

Microprocessor and Microcontroller – SPH1313

8086 MICROPROCESSOR

FEATURES OF 8086 MICROPROCESSOR

1. It is 16-bit microprocessor
2. It has a 16-bit data bus, so it can read data from or write data to memory and ports either 16-bit or 8-bit at a time.
3. It has 20 bit address bus and can access up to 2^{20} memory locations (1 MB).
4. It can support up to 64K I/O ports
5. It provides 14, 16-bit registers
6. It has multiplexed address and data bus AD_0-AD_{15} & A_{16}-A_{19}
7. It requires single phase clock with 33% duty cycle to provide internal timing.
8. Prefetches up to 6 instruction bytes from memory and queues them in order to speed up the processing.
9. 8086 supports 2 modes of operation

a. Minimum mode

b. Maximum mode

Architecture of 8086 microprocessor:

- As shown in the below figure. 1, the 8086 CPU is divided into two independent functional parts
  - Bus Interface Unit(BIU)
  - Execution Unit(EU)
- Dividing the work between these two units' speeds up processing.

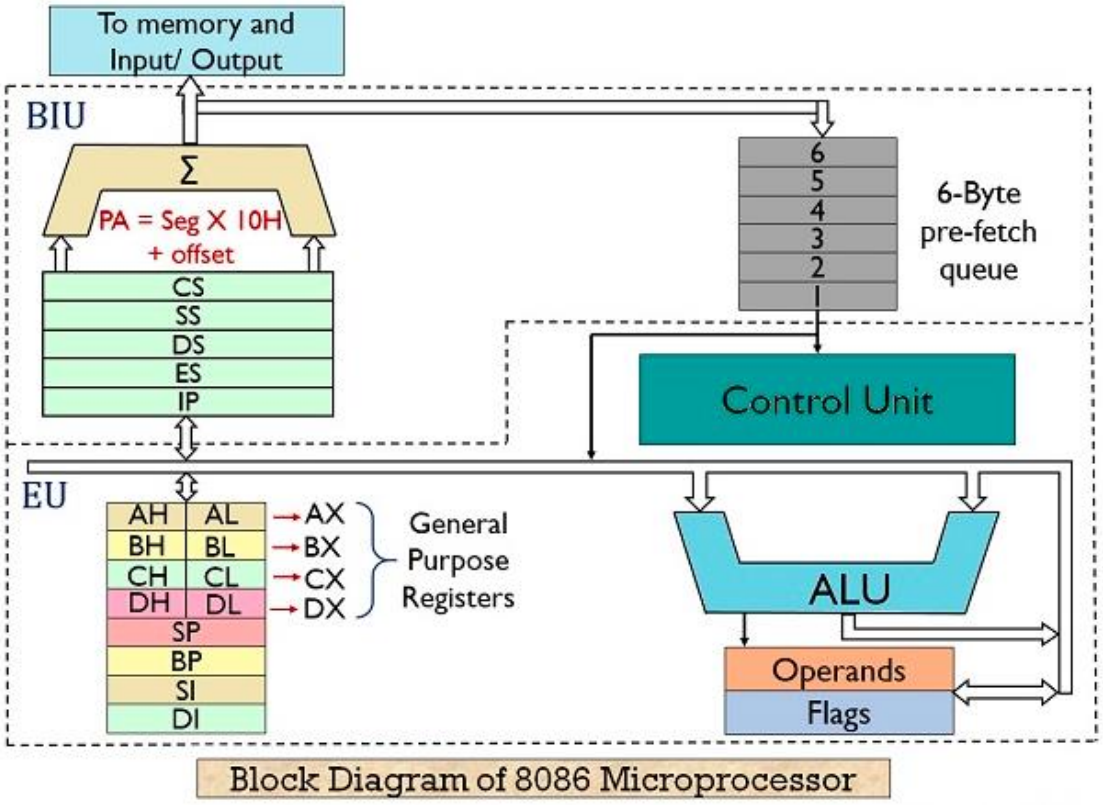


figure. 1 Internal Architevture of 8086 Microprocessor

The Execution Unit (EU):

- The execution unit of the 8086 tells the BIU where to fetch instructions or data from, decodes instructions, and executes instructions.
- The EU contains **control circuitry,** which directs internal operations.
- A decoder in the EU translates instructions fetched from memory into a series of actions, which the EU carries out.
- The EU has a 16-bit **arithmetic logic unit** (ALU) which can add, subtract, AND, OR, XOR, increment, decrement, complement or shift binary numbers.
- The main functions of EU are:
    - Decoding of Instructions
    - Execution of instructions
        - ✓ Steps
            - EU extracts instructions from top of queue in BIU
            - Decode the instructions

- Generates operands if necessary
- Passes operands to BIU & requests it to perform read or write bus cycles to memory or I/O
- Perform the operation specified by the instruction on operands

Bus Interface Unit (BIU):

- The BIU sends out addresses, fetches instructions from memory, reads data from ports and memory, and writes data to ports and memory.
- In simple words, the BIU handles all transfers of data and addresses on the buses for the execution unit.

8086 HAS PIPELINING ARCHITECTURE:

- While the EU is decoding an instruction or executing an instruction, which does not require use of the buses, the BIU fetches up to six instruction bytes for the following instructions.
- The BIU stores these pre-fetched bytes in a first-in-first-out register set called a *queue.*
- When the EU is ready for its next instruction from the queue in the BIU.
- Except in the case of JMP and CALL instructions, where the queue must be dumped and then reloaded starting from a new address, this pre-fetch and queue scheme greatly speeds up processing.
- Fetching the next instruction while the current instruction executes is called **pipelining**.

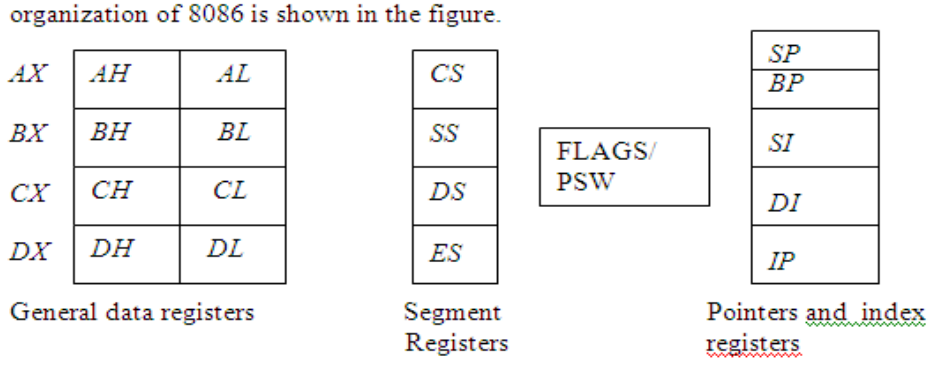
Register organization:

- 8086 has a powerful set of registers known as *general purpose registers* and *special purpose registers.*
- All of them are 16-bit registers.
- *General purpose registers:*
  - These registers can be used as either 8-bit registers or 16-bit registers.
  - They may be either used for holding data, variables and intermediate

results temporarily or for other purposes like a counter or for storing offset address for some particular addressing modes etc.

- **Special purpose registers:**
  - These registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes.
- The 8086 registers are classified into the following types:
  - General Data Registers
  - Segment Registers
  - Pointers and Index Registers
  - Flag Register

The register set of 8086 can be categorized into 4 different groups. The register organization of 8086 is shown in the figure.



Register organization of 8086

General Data Registers:

- The registers *AX*, *BX*, *CX* and *DX* are the general purpose 16-bit registers.
- *AX* is used as 16-bit accumulator. The lower 8-bit is designated as *AL* and higher 8-bit is designated as *AH*. *AL*

  can be used as an 8-bit accumulator for 8-bit operation.
- All data register can be used as either 16 bit or 8 bit. *BX* is a 16 bit register, but *BL* indicates the lower 8-bit of

  BX and *BH* indicates the higher 8-bit of *BX*.
- The register *BX* is used as offset storage for forming physical address in case of certain addressing modes.
- The register *CX* is used default counter in case of string and loop instructions.

- *DX* register is a general purpose register which may be used as an implicit operand or destination in case of a few instructions.

Segment Registers:

- There are 4 segment registers. They are:
  - Code Segment Register(CS)
  - Data Segment Register(DS)
  - Extra Segment Register(ES)
  - Stack Segment Register(SS)

- The 8086 architecture uses the concept of **segmented memory**. 8086 able to address a memory capacity of 1 megabyte and it is byte organized. This 1 megabyte memory is divided into 16 logical segments. Each segment contains 64 kbytes of memory.
- Code segment register (CS): is used for addressing memory location in the code segment of the memory, where the executable program is stored.
- Data segment register (DS): points to the data segment of the memory where the data is stored.
- Extra Segment Register (ES) : also refers to a segment in the memory which is another data segment in the memory.
- Stack Segment Register (SS): is used for addressing stack segment of the memory. The stack segment is that segment of memory which is used to store stack data.
- While addressing any location in the memory bank, the **physical address** is calculated from two parts:

  Physical address= segment address + offset address

- The first is segment address, the segment registers contain 16-bit segment base addresses, related to different segment.
- The second part is the offset value in that segment.

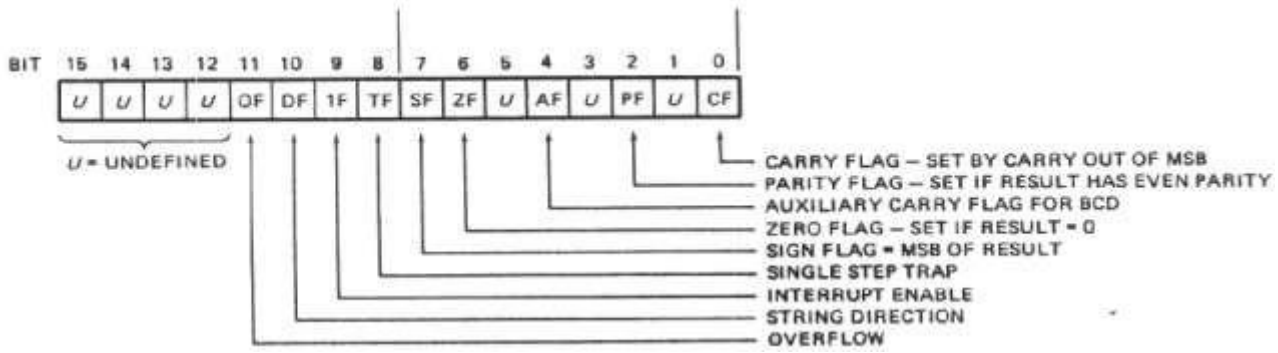
Pointers and Index Registers:

- The index and pointer registers are given below:
    - IP—Instruction pointer-store memory location of next instruction to be executed
    - BP—Base pointer
    - SP—Stack pointer
    - SI—Source index
    - DI—Destination index
- The pointers registers contain offset within the particular segments.
    - The pointer register *IP* contains offset within the code segment.
    - The pointer register *BP* contains offset within the data segment.
    - Thee pointer register *SP* contains offset within the stack segment.
- The index registers are used as general purpose registers as well as for offset storage in case of indexed, base indexed and relative base indexed addressing modes.
- The register *SI* is used to store the offset of source data in data segment.
- The register *DI* is used to store the offset of destination in data or extra segment.
- The index registers are particularly useful for string manipulation.

8086 flag register and its functions:

- The 8086 flag register contents indicate the results of computation in the *ALU*. It also contains some flag bits to control the *CPU* operations.
- A 16 bit flag register is used in 8086. It is divided into two parts .
  - Condition code or status flags
  - Machine control flags
- The **condition code flag register** is the lower byte of the 16-bit flag register. The condition code flag register is identical to 8085 flag register, with an additional overflow flag.
- The **control flag register** is the higher byte of the flag register. It contains three flags namely direction flag (*D*), interrupt flag (*I*) and trap flag (*T*).

Flag register configuration



The description of each flag bit is as follows:

***SF*- Sign Flag:** This flag is set, when the result of any computation is negative. For signed computations the sign flag equals the MSB of the result.

***ZF*- Zero Flag:** This flag is set, if the result of the computation or comparison performed by the previous instruction is zero.

***PF*- Parity Flag:** This flag is set to 1, if the lower byte of the result contains even number of 1's.

***CF*- Carry Flag:** This flag is set, when there is a carry out of MSB in case of addition or a borrow in case of subtraction.

***AF*-Auxilary Carry Flag:** This is set, if there is a carry from the lowest nibble, i.e, bit three during addition, or borrow for the lowest nibble, i.e, bit three, during subtraction.

***OF*- Over flow Flag:** This flag is set, if an overflow occurs, i.e, if the result of a signed operation is large enough to accommodate in a destination register. The result is of more than 7-bits in size in case of 8-bit signed operation and more than 15-bits in size in case of 16-bit sign operations, and then the overflow will be set.

***TF*- Tarp Flag:** If this flag is set, the processor enters the single step execution mode. The processor executes the current instruction and the control is transferred to the Trap interrupt service routine.

***IF*- Interrupt Flag:** If this flag is set, the mask able interrupts are recognized by the CPU, otherwise they are ignored.

***D*- Direction Flag:** This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address, i.e., auto incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e., auto decrementing mode.

Memory Segmentation

- The memory in an 8086 based system is organized as segmented memory.
- The CPU 8086 is able to access 1MB of physical memory. The complete 1MB of memory can be divided into 16 segments, each of 64KB size and is addressed by one of the segment register.
- The 16-bit contents of the segment register actually point to the starting location of a particular segment. The address of the segments may be assigned as 0000H to F000h respectively.
- To address a specific memory location within a segment, we need an offset address. The offset address values are from 0000H to FFFFH so that the physical addresses range from 00000H to FFFFFH

Physical address is calculated as below:

Ex: Segment

address ---------- □ 1005H Offset address ----------□ 5555H

Segment address -------□ 1005H-0001 0000 0000 0101

Shifted left by 4 Positions 0001 0000 0000 0101 0000

+

Offset address --- 5555H ------ 0101 0101 0101 0101

Physical address = Segment address * 10H + Offset address.

Physical address -------155A5H0001 0101 0101 1010 0101

The main advantages of the segmented memory scheme are as follows:

1. Allows the memory capacity to be 1MB although the actual addresses to be handled are of 16-bit size.
2. Allows the placing of code, data and stack portions of the same program in different parts (segments) of memory, for data and code protection.
3. Permits a program and/or its data to be put into different areas of

memory each time the program is executed, i.e., provision for relocation is done.

Overlapping and Non-overlapping Memory segments:

- In the overlapping area locations physical address = CS1+IP1 = CS2+IP2. Where '+' indicates the procedure of physical address formation.

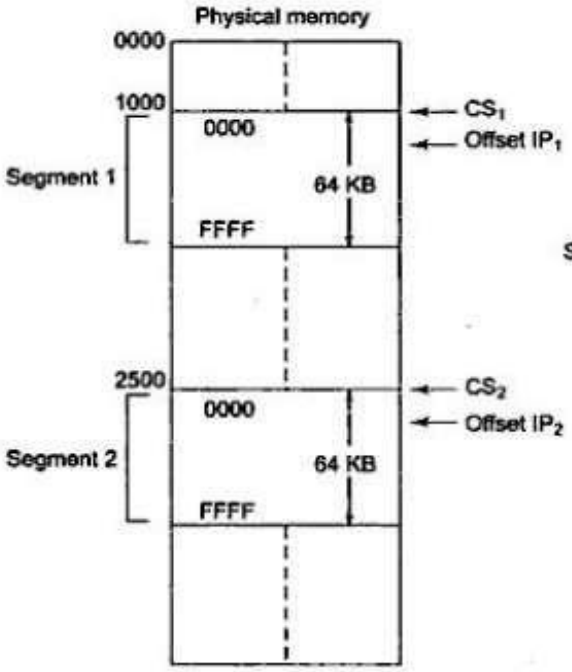


Fig. 1.3(a) *Non-overlapping Segments*

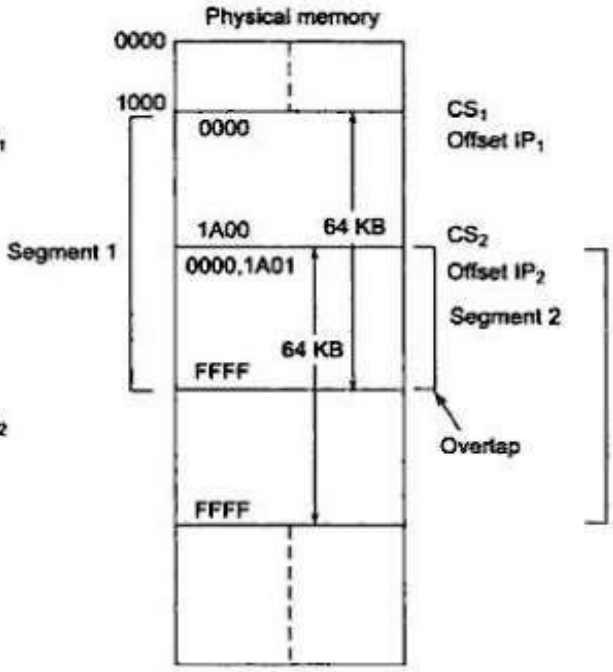


Fig. 1.3(b) *Overlapping Segments*

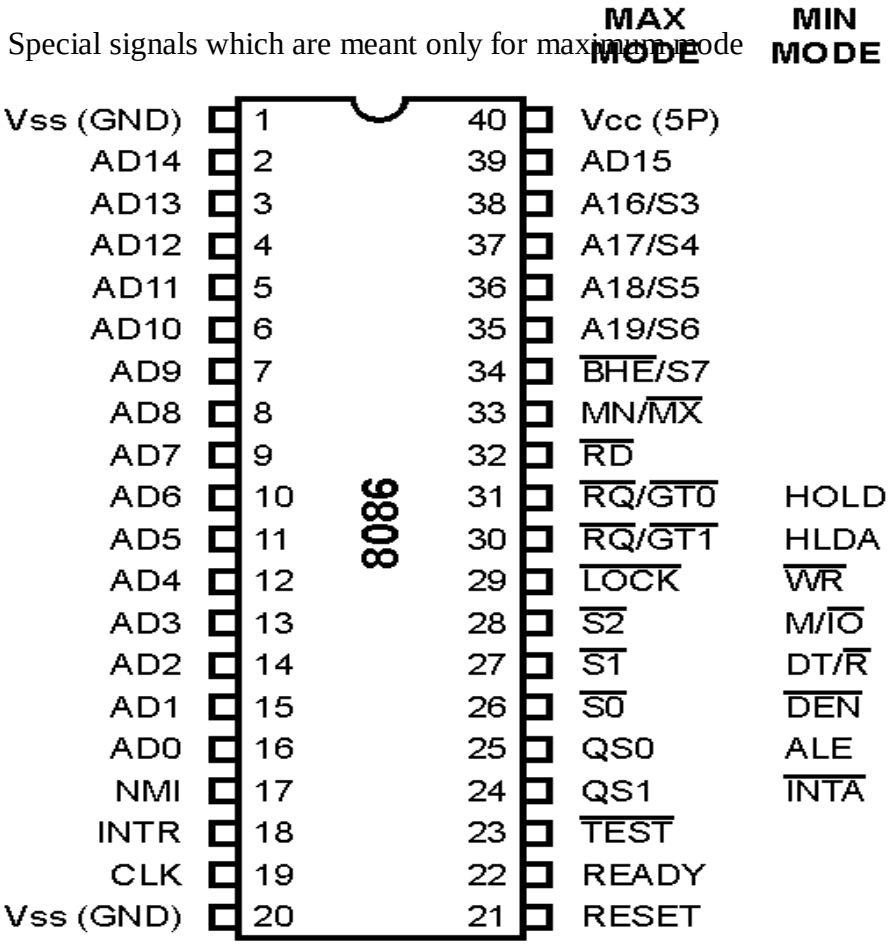
Pin Diagram of 8086:

Signal description of 8086:

- The 8086 is a 16-bit microprocessor. This microprocessor operates in single processor or multiprocessor configurations to achieve high performance.
- The pin configuration of 8086 is shown in the figure. Some of the pins serve a particular function in minimum mode (single processor mode) and others function in maximum mode (multiprocessor mode).

The 8086 signals are categorized into 3 types:

1. Common signals for both minimum mode and maximum mode.
2. Special signals which are meant only for minimum mode
3. Special signals which are meant only for maximum mode



Common Signals for both Minimum mode and Maximum mode:

AD_7 □ AD_0 : The address/ data bus lines are the multiplexed address data bus and contain the right most eight bit of memory address or data. The address and data bits are separated by using *ALE* signal.

AD_{15} □ AD_8 : The address/data bus lines compose the upper multiplexed address/data bus. This lines contain address A_{15} □ A_8 or data bus D_{15} □ D_8 . The address and data bits are separated by using *ALE* signal.

A_{19} / S_6 □ A_{18} / S_3 The address/status bus bits are multiplexed to provide address signals A_{19} □ A_{16} and also status bits S_6 □ S_3 . The address bits are separated from the status bits using the ALE signals. The status bit S_6 is always a logic 0, bit S_5 indicates the condition of the interrupt flag bit. The S_4 and S_3 b used for memory access. indicate which segment register is presently

S_4	S_3	Type of segment register used
0	0	Extra segment
0	1	Stack segment
1	0	Code or no segment
1	1	Data Segment

BHE / S_7

The bus high enable (BHE) signal is used to indicate the transfer of data over the higher order □D_{15} □ D_8 □ data bus. It goes low for the data transfer over D_{15} □ D_8 and is used to derive chip select of odd address memory bank or peripherals.

RD : Read: whenever the read signal is at logic 0, the data bus receives the data from the memory or I/O devices connected to the system

READY: This is the acknowledgement from the slow devices or memory that they have completed the data transfer operation. This signal is active high.

INTR: Interrupt Request: Interrupt request is used to request a hardware interrupt of *INTR* is held high when interrupt enable flag is set, the 8086 enters an interrupt acknowledgement cycle after the current instruction has completed its execution.

TEST : This input is tested by "*WAIT*" instruction. If the *TEST* input goes low; execution will continue. Else the

processor remains in an idle state.

NMI- Non-maskable Interrupt: The non-maskable interrupt input is similar to *INTR* except that the *NMI* interrupt does not check for interrupt enable flag is at logic 1, i.e, *NMI* is not maskable internally by software. If *NMI* is activated, the interrupt input uses interrupt vector 2.

RESET: The reset input causes the microprocessor to reset itself. When 8086 reset, it restarts the execution from memory location *FFFF0H*. The reset signal is active high and must be active for at least four clock cycles.

CLK: Clock input: The clock input signal provides the basic timing input signal for processor and bus control operation. It is asymmetric square wave with 33% duty cycle.

V_{CC} (+5V): Power supply for the operation of the internal circuit

GND: Ground for the internal circuit

MN / *MX* : The minimum/maximum mode signal to select the mode of operation either in minimum or maximum mode configuration. Logic 1 indicates minimum mode.

Minimum mode Signals: The following signals are for minimum mode operation of 8086.

M / IO- Memory/*IO* *M / IO* signal selects either memory operation or I/O operation. This line indicates that the microprocessor address bus contains either a memory address or an I/O port address. Signal high at this pin indicates a memory operation. This line is logically equivalent to S_2 in maximum mode.

INTA - Interrupt acknowledge: The interrupt acknowledge signal is a response to the INTR input signal. The

INTA- signal is normally used to gate the interrupt vector number onto the data bus in response to an interrupt request.

ALE- Address Latch Enable: This output signal indicates the availability of valid address on the address/data bus, and is connected to latch enable input of latches.

DT / R : Data transmit/Receive: This output signal is used to decide the direction of date flow through the bi-

directional buffer. *DT / R* □ 1 Indicates transmitting and *DT / R* □ 0 indicates receiving the data.

DEN Data Enable: Data bus enable signal indicates the availability of valid data over the address/data lines.

WR: Wrrite: whenever the write signal is at logic 0, the data bus transmits the data to the memory or I/O devices connected to the system.

HOLD: The hold input request a direct memory access (DMA). If the hold signal is at logic 1, the micro process stops its normal execution and places its address, data and control bus at the high impedance state.

HLDA: Hold acknowledgement indicates that 8086 has entered into the hold state

.

– – **Maximum mode signal: The following signals are for maximum mode operation of 8086.**

S_2 , S_1, S_0 - Status lines: These are the status lines that reflect the type of operation being carried out by the processor.

These status lines are encoded as follows

S_2	S_1	S_0	Function
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive (In active)

$\overline{LOCK}$: The lock output is used to lock peripherals off the system, i.e, the other system bus masters will be prevented from gaining the system bus.

QS_1	QS_0	Function
0	0	No operation, queue is idle
0	1	First byte of opcode
1	0	Queue is empty
1	1	Subsequent byte of opcode

QS_1 and QS_0 - Queue status: The queue status bits shows the status of the internal instruction queue. The encoding of these signals is as follows

$\overline{RQ}/\overline{GT}1$ and $\overline{RQ}/\overline{GT}0$ - request/Grant: The request/grant pins are used by other local bus masters to force the processor to release the local bus at the end of the processors current bus cycle. These lines are bi- directional and are used to both request and grant a *DMA* operation. $\overline{RQ}/\overline{GT}0$ is having higher priority than $\overline{RQ}/\overline{GT}1$

Addressing modes of 8086

- Addressing mode indicates a way of locating data or operands.
- The addressing modes describe the types of operands and the way they are accessed for executing an instruction.

The addressing modes for sequential control transfer instructions are:

1. **Immediate:** In this type of addressing, immediate data is a part of instruction and appears in the form of successive byte or bytes.

   Ex: MOV AX, 0005H

   In the above example, 0005H is the immediate data. The immediate data may be 8-bit or 16-bit in size.

2. **Direct:** In the direct addressing mode a 16-bit memory address (offset) is directly specified in the instruction as a part of it.

   Ex: MOV AX, [5000H]

   Here, data resides in a memory location in the data segment, whose effective address may be completed using 5000H as the offset address and content of DS as segment address. The effective address here, is 10H * DS + 5000H.

3. **Register:** In register addressing mode, the data is stored in a register and is referred using the particular register. All the registers, except IP, may be used in this mode.

   Ex: MOV BX, AX

4. **Register Indirect:** Sometimes, the address of the memory location, which contains data or operand, is determined in an indirect way, using the offset register. This mode of addressing is known as register indirect mode. In this addressing mode, the offset address of data is in either BX or SI or DI register. The default segment is either DS or ES. The data is supposed to be available at the address pointed to by the content of any of the above registers in the default data segment.

Ex: MOV AX, [BX]

Here, data is present in a memory location in DS whose offset address is in BX. The effective address of the data is given as 10H * DS+[BX].

5. **Indexed:** In this addressing mode, offset of the operand is stored in one of the index registers. DS and ES are the default segments for index registers, SI and DI respectively. This is a special case of register indirect addressing mode.

Ex: MOV AX, [SI]

Here, data is available at an offset address stored in SI in DS. The effective address, in this case, is computed as 10*DS+[SI].

6. **Register Relative:** In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI and DI in the default (either DS or ES) segment.

Ex: MOV AX, 50H[BX]

Here, the effective address is given as 10H *DS+50H+[BX]

7. **Based Indexed:** The effective address of data is formed, in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

Ex: MOV AX, [BX][SI]

Here, BX is the base register and SI is the index register the effective address is computed as 10H * DS + [BX] + [SI].

8. **Relative Based Indexed:** The effective address is formed by adding an 8 or 16-bit displacement with the sum of the contents of any one of the base register (BX or BP) and any one of the index register, in a default segment.

Ex: MOV AX, 50H [BX] [SI]

Here, 50H is an immediate displacement, BX is base register and SI is an index register the effective address of data is computed as

10H * DS + [BX] + [SI] + 50H

9. Implied addressing mode:

The Instruction using this mode have no operands. For example:

CLC which clears carry flag to zero.

INSTRUCTION SET OF 8086

The 8086 instructions are categorized into the following main types.

1. Data Copy / Transfer Instructions
2. Arithmetic and Logical Instructions
3. Shift and Rotate Instructions
4. Loop Instructions
5. Branch Instructions
6. String Instructions
7. Flag Manipulation Instructions
8. Machine Control Instructions

1 DATA COPY / TRANSFER INSTRUCTIONS:

MOV:

This instruction copies a word or a byte of data from some source to a destination. The destination can be a register or a memory location. The source can be a register, a memory location, or an immediate number.

MOV AX, BX
MOV AX, 5000H
MOV AX, [2000H]
MOV AX, 50H[BX]
MOV [734AH], BX
MOV DS, CX
MOV CL, [357AH]

Direct loading of the segment registers with immediate data is not permitted.

PUSH: Push to Stack

This instruction pushes the contents of the specified register/memory location on to the stack. The stack pointer is decremented by 2, after each execution of the instruction.

E.g. PUSH AX

• PUSH DS

• PUSH [5000H]

POP: Pop from Stack

This instruction when executed, loads the specified register/memory location with the contents of the memory location of which the address is formed using the current stack segment and stack pointer.

The stack pointer is incremented by 2 Eg. POP AX

POP DS POP [5000H]

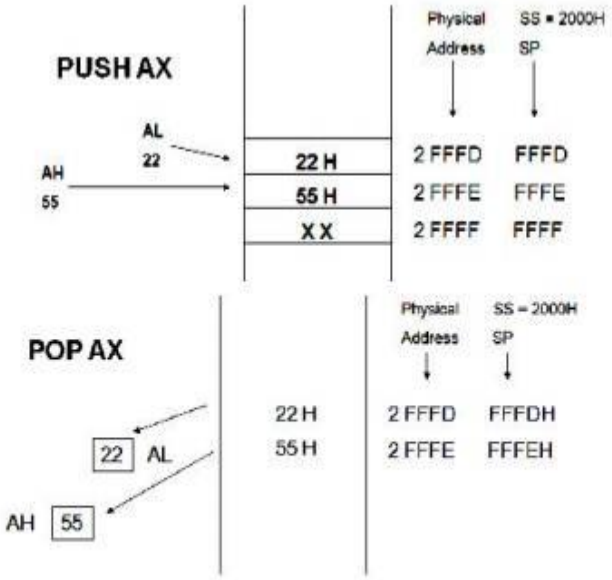


Fig 1.8 Push into and Popping Register Content from Stack Memory

XCHG: Exchange byte or word

This instruction exchange the contents of the specified source and destination operands

Eg. XCHG [5000H], AX

XCHG BX, AX

Input and output port transfer instructions:

IN:

Copy a byte or word from specified port to accumulator.

Eg. IN AL,03H

IN AX,DX

OUT:

Copy a byte or word from accumulator specified port.

Eg. OUT 03H, AL

OUT DX, AX

XLAT:

Translate byte using look-up table

Eg. LEA BX, TABLE1

MOV AL, 04H

XLAT

LEA:

Load effective address of operand in specified register. [reg] offset portion of address in DS

Eg. LEA reg, offset

LDS:

Load DS register and other specified register from memory. [reg] [mem] [DS] [mem + 2] Eg. LDS reg, mem

LES:

Load ES register and other specified register from memory. [reg] [mem]

[ES] [mem + 2] Eg. LES reg, mem

Flag transfer instructions:

LAHF:

Load (copy to) AH with the low byte the flag register. [AH] [Flags low byte]
Eg. LAHF

SAHF:

Store (copy) AH register to low byte of flag register. [Flags low byte] [AH]
Eg. SAHF

PUSHF:

Copy flag register to top of stack. [SP] [SP] – 2
[[SP]] [Flags] Eg. PUSHF

POPF:

Copy word at top of stack to flag register. [Flags] [[SP]]
[SP] [SP] + 2

2 ARITHMETIC INSTRUCTIONS:

The 8086 provides many arithmetic operations: addition, subtraction, negation, multiplication and comparing two values.

ADD:

The add instruction adds the contents of the source operand to the destination operand. All the flags are modified.

Eg. ADD AX, 0100H
ADD AX, BX
ADD AX, [SI]
ADD AX, [5000H]
ADD [5000H], 0100H
ADD 0100H

ADC: Add with Carry

This instruction performs the same operation as ADD instruction, but adds the carry flag to the result. All the flags are modified.

Eg. ADC 0100H
ADC AX, BX
ADC AX, [SI]
ADC AX, [5000]
ADC [5000],
ADC 0100H

SUB: Subtract

The subtract instruction subtracts the source operand from the destination operand and the result is left in the destination operand.
All the flags are modified.

Eg. SUB AX, 0100H
SUB AX, BX
SUB AX, [5000H]
SUB [5000H], 0100H

SBB: Subtract with Borrow

The subtract with borrow instruction subtracts the source operand and the borrow flag (CF) which may reflect the result of the previous calculations, from the destination operand. All the flags are modified.

Eg. SBB AX, 0100H
SBB AX, BX
SBB AX, [5000H]
SBB [5000H], 0100H

INC: Increment

This instruction increases the contents of the specified Register or memory location by 1. Immediate data cannot be operand of this instruction.

Eg. INC AX
INC [BX]
INC [5000H]

DEC: Decrement

The decrement instruction subtracts 1 from the contents of the specified register or memory location.
Eg. DEC AX
DEC [5000H]

NEG: Negate

The negate instruction forms 2's complement of the specified destination in the instruction. The destination can be a register or a memory location. This instruction can be implemented by inverting each bit and adding 1 to it.
Eg. NEG AL
AL = 0011 0101 35H Replace number in AL with its 2's complement
AL = 1100 1011 = CBH

CMP: Compare

This instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location.

S=D ; Z=1
S>D; C=1
S<D; C=0

Eg. CMP BX, 0100H
CMP AX, 0100H
CMP [5000H], 0100H
CMP BX, [SI]
CMP BX, CX

MUL:Unsigned Multiplication Byte or Word

This instruction multiplies an unsigned byte or word by the contents of AL. Eg. MUL BH; (AX) (AL) x (BH)

MUL CX; (DX)(AX) (AX) x (CX)
MUL WORD PTR [SI]; (DX)(AX) (AX) x ([SI])

IMUL:Signed Multiplication

This instruction multiplies a signed byte in source operand by a signed byte in AL or a signed word in source operand by a signed word in AX.

Eg. IMUL BH
IMUL CX
IMUL [SI]

CBW: Convert Signed Byte to Word

This instruction copies the sign of a byte in AL to all the bits in AH. AH is then said to be sign extension of AL.

Eg. CBW

AX= 0000 0000 1001 1000 Convert signed byte in AL signed word in AX. Result in AX = 1111 1111 1001 1000

CWD: Convert Signed Word to Double Word

This instruction copies the sign of a byte in AL to all the bits in AH. AH is then said to be sign extension of AL.

Eg. CWD

Convert signed word in AX to signed double word in DX: AX DX= 1111 1111 1111 1111

Result in AX = 1111 0000 1100 0001

DIV: Unsigned division

This instruction is used to divide an unsigned word by a byte or to divide an unsigned double word by a word.

Eg. DIV CL; Word in AX / byte in CL; Quotient in AL, remainder in AH
DIV CX; Double word in DX and AX / word; in CX, and Quotient in AX; remainder in
DX

AAA: ASCII Adjust After Addition

The AAA instruction is executed after an ADD instruction that adds two ASCII coded operand to give a byte of result in AL. The AAA instruction converts the resulting contents of Al to a unpacked decimal digits.

Eg. ADD CL, DL; [CL] = 32H = ASCII for 2; [DL] = 35H = ASCII for 5; Result [CL] = 67H

MOV AL, CL; Move ASCII result into AL since; AAA adjust only [AL] AAA; [AL]=07, unpacked BCD for 7

AAS: ASCII Adjust AL after Subtraction

This instruction corrects the result in AL register after subtracting two unpacked ASCII operands. The result is in unpacked decimal format. The procedure is similar to AAA instruction except for the subtraction of 06 from AL.

AAM: ASCII Adjust after Multiplication

This instruction, after execution, converts the product available In AL into unpacked BCD format.

Eg. MOV AL, 04; AL = 04 MOV BL ,09; BL = 09

MUL BL; AX = AL*BL; AX=24H AAM; AH = 03, AL=06

AAD: ASCII Adjust before Division

This instruction converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL. This adjustment must be made before dividing the two unpacked BCD digits in AX by an unpacked BCD byte. In the instruction sequence, this instruction appears Before DIV instruction.

Eg. AX 05 08
AAD result in AL 00 3A 58D = 3A H in AL

The result of AAD execution will give the hexadecimal number 3A in AL and 00 in AH where 3A is the hexadecimal Equivalent of 58 (decimal).

DAA: Decimal Adjust Accumulator

This instruction is used to convert the result of the addition of two packed BCD numbers to a valid BCD number. The result has to be only in AL.

Eg. AL = 53 CL = 29

ADD AL, CL; AL (AL) + (CL); AL 53 + 29; AL 7C

DAA; AL 7C + 06 (as C>9); AL 82

DAS: Decimal Adjust after Subtraction

This instruction converts the result of the subtraction of two packed BCD numbers to a valid BCD number. The subtraction has to be in AL only.

Eg. AL = 75, BH = 46

SUB AL, BH; AL 2 F = (AL) - (BH) ; AF = 1

DAS; AL 2 9 (as F>9, F - 6 = 9)

LOGICAL INSTRUCTIONS

AND: Logical AND

This instruction bit by bit ANDs the source operand that may be an immediate register or a memory location to the destination operand that may a register or a memory location. The result is stored in the destination operand.
Eg. AND AX, 0008H
AND AX, BX

OR: Logical OR

This instruction bit by bit ORs the source operand that may be an immediate, register or a memory location to the destination operand that may a register or a memory location. The result is stored in the destination operand.
Eg. OR AX, 0008H
OR AX, BX

NOT: Logical Invert

This instruction complements the contents of an operand register or a memory location, bit by bit.

Eg. NOT AX
NOT [5000H]

OR: Logical Exclusive OR

This instruction bit by bit XORs the source operand that may be an immediate, register or a memory location to the destination operand that may a register or a memory location. The result is stored in the destination operand.
Eg. XOR AX, 0098H
XOR AX, BX

TEST: Logical Compare Instruction

The TEST instruction performs a bit by bit logical AND operation on the two operands. The result of this ANDing operation is not available for further use, but flags are affected.
Eg. TEST AX, BX
TEST [0500], 06H

3 Shift and Rotate Instructions

SAL/SHL: SAL / SHL destination, count.

SAL and SHL are two mnemonics for the same instruction. This instruction shifts each bit in the specified destination to the left and 0 is stored at LSB position. The MSB is shifted into the carry flag. The destination can be a byte or a word. It can be in a register or in a memory location. The number of shifts is indicated by count.

Eg. SAL CX, 1
SAL AX, CL

SHR: SHR destination, count

This instruction shifts each bit in the specified destination to the right and 0 is stored at MSB position. The LSB is shifted into the carry flag. The destination can be a byte or a word.

It can be a register or in a memory location. The number of shifts is indicated by count. Eg. SHR CX, 1

MOV CL, 05H
SHR AX, CL

SAR: SAR destination, count

This instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted out of the MSB position, a copy of the old MSB is put in the MSB position. The LSB will be shifted into CF.
Eg. SAR BL, 1
MOV CL, 04H
SAR DX, CL

ROL Instruction: ROL destination, count

This instruction rotates all bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF.

Eg. ROL CX, 1
MOV CL, 03H
ROL BL, CL

ROR Instruction: ROR destination, count

This instruction rotates all bits in a specified byte or word to the right some number of bit positions. LSB is placed as a new MSB and a new CF.

Eg. ROR CX, 1
MOV CL, 03H
ROR BL, CL

RCL Instruction: RCL destination, count

This instruction rotates all bits in a specified byte or word some number of bit positions to the left along with the carry flag. MSB is placed as a new carry and previous carry is place as new LSB.
Eg. RCL CX, 1
MOV CL, 04H
RCL AL, CL

RCR Instruction: RCR destination, count

This instruction rotates all bits in a specified byte or word some number of bit positions to the right *along with the carry flag*. LSB is placed as a new carry and previous carry is place as new MSB.
Eg. RCR CX, 1
MOV CL, 04H
RCR AL, CL

ROR Instruction: ROR destination, count

This instruction rotates all bits in a specified byte or word to the *right* some number of bit positions. LSB is placed as a new MSB and a new CF.

Eg. ROR CX, 1
MOV CL, 03H
ROR BL, CL

RCL Instruction: RCL destination, count

This instruction rotates all bits in a specified byte or word some number of bit positions to the left along with the carry flag. MSB is placed as a new carry and previous carry is place as new LSB.

Eg. RCL CX, 1
MOV CL, 04H
RCL AL, CL

RCR Instruction: RCR destination, count

This instruction rotates all bits in a specified byte or word some number of bit positions to the right *along with the carry flag*. LSB is placed as a new carry and previous carry is place as new MSB.

Eg. RCR CX, 1
MOV CL, 04H
RCR AL, CL

4 LOOP INSTRUCTIONS:

Unconditional LOOP Instructions

LOOP: LOOP Unconditionally

This instruction executes the part of the program from the Label or address specified in the instruction upto the LOOP instruction CX number of times. At each iteration, CX is decremented automatically and JUMP IF NOT ZERO structure.
Example: MOV CX, 0004H

Conditional LOOP Instructions

LOOPZ / LOOPE Label

Loop through a sequence of instructions from label while ZF=1 and CX=0.

LOOPNZ / LOOPENE Label

Loop through a sequence of instructions from label while ZF=1 and CX=0.

5 Branch Instructions:

Branch Instructions transfers the flow of execution of the program to a new address specified in the instruction directly or indirectly. When this type of instruction is executed, the CS and IP registers get loaded with new values of CS and IP corresponding to the location to be transferred.

The Branch Instructions are classified into two types

1. Unconditional Branch Instructions.
2. Conditional Branch Instructions.

1.4.5.1 Unconditional Branch Instructions:

In Unconditional control transfer instructions, the execution control is transferred to the specified location independent of any status or condition. The CS and IP are unconditionally modified to the new CS and IP.

CALL: Unconditional Call

This instruction is used to call a Subroutine (Procedure) from a main program. Address of procedure may be specified directly or indirectly. There are two types of procedure depending upon whether it is available in the same segment or in another segment.

i. Near CALL i.e., ±32K displacement.

ii. For CALL i.e., anywhere outside the segment.

On execution this instruction stores the incremented IP & CS onto the stack and loads the CS & IP registers with segment and offset addresses of the procedure to be called.

RET: Return from the Procedure.

At the end of the procedure, the RET instruction must be executed. When it is executed, the previously stored content of IP and CS along with Flags are retrieved into the CS, IP and Flag registers from the stack and execution of the main program continues further.

INT N: Interrupt Type N.

In the interrupt structure of 8086, 256 interrupts are defined corresponding to the types from 00H to FFH. When INT N instruction is executed, the type byte N is multiplied by 4 and the contents of IP and CS of the interrupt service routine will be taken from memory block in 0000 segment.

INTO: Interrupt on Overflow

This instruction is executed, when the overflow flag OF is set. This is equivalent to a Type 4 Interrupt instruction.

JMP: Unconditional Jump

This instruction unconditionally transfers the control of execution to the specified address using an 8-bit or 16-bit displacement. No Flags are affected by this instruction.

IRET: Return from ISR

When it is executed, the values of IP, CS and Flags are retrieved from the stack to continue the execution of the main program.

MOV BX, 7526H

Label 1 MOV AX, CODE

OR BX, AX LOOP Label 1

Conditional Branch Instructions

When this instruction is executed, execution control is transferred to the address specified relatively in the instruction, provided the condition implicit in the Opcode is satisfied. Otherwise execution continues sequentially.

JZ/JE Label

Transfer execution control to address ‘Label’, if ZF=1.

JNZ/JNE Label

Transfer execution control to address ‘Label’, if ZF=0

JS Label

Transfer execution control to address ‘Label’, if SF=1.

JNS Label

Transfer execution control to address ‘Label’, if SF=0.

JO Label

Transfer execution control to address ‘Label’, if OF=1.

JNO Label

Transfer execution control to address ‘Label’, if OF=0.

JNP Label

Transfer execution control to address ‘Label’, if PF=0.

JP Label

Transfer execution control to address ‘Label’, if PF=1.

JB Label

Transfer execution control to address ‘Label’, if CF=1.

JNB Label

Transfer execution control to address ‘Label’, if CF=0.

JCXZ Label

Transfer execution control to address ‘Label’, if CX=0

6 STRING MANIPULATION INSTRUCTIONS

A series of data byte or word available in memory at consecutive locations, to be referred as Byte String or Word String. A String of characters may be located in consecutive memory locations, where each character may be represented by its ASCII equivalent. The 8086 supports a set of more powerful instructions for string manipulations for referring to a string, two parameters are required.

I. Starting and End Address of the String.

II. Length of the String.

The length of the string is usually stored as count in the CX register. The incrementing or decrementing of the pointer, in string instructions, depends upon the Direction Flag (DF) Status. If it is a Byte string operation, the index registers are updated by one. On the other hand, if it is a word string operation, the index registers are updated by two.

REP: Repeat Instruction Prefix

This instruction is used as a prefix to other instructions, the instruction to which the REP prefix is provided, is executed repeatedly until the CX register becomes zero (at each iteration CX is automatically decremented by one).

i. REPE / REPZ - repeat operation while equal / zero.

ii. REPNE / REPNZ - repeat operation while not equal / not zero. These are used for CMPS, SCAS instructions only, as instruction prefixes.

MOVSB / MOVSW: Move String Byte or String Word

Suppose a string of bytes stored in a set of consecutive memory locations is to be moved to another set of destination locations. The starting byte of source string is located in the memory location whose address may be computed using SI (Source Index) and DS (Data Segment) contents. The starting address of the destination locations where this string has to be relocated is given by DI (Destination Index) and ES (Extra Segment) contents.

CMPS: Compare String Byte or String Word

The CMPS instruction can be used to compare two strings of byte or words. The length of the string must be stored in the register CX. If both the byte or word strings are equal, zero Flag is set.

The REP instruction Prefix is used to repeat the operation till CX (counter) becomes zero or the condition specified by the REP Prefix is False.

SCAN: Scan String Byte or String Word

This instruction scans a string of bytes or words for an operand byte or word specified in the register AL or AX. The String is pointed to by ES: DI register pair. The length of the string s stored in CX. The DF controls the mode for scanning of the string. Whenever a match to the specified operand is found in the string, execution stops and the zero Flag is set. If no match is found, the zero flag is reset.

LODS: Load String Byte or String Word

The LODS instruction loads the AL / AX register by the content of a string pointed to by DS: SI register pair. The SI is modified automatically depending upon DF, If it is a byte transfer (LODSB), the SI is modified by one and if it is a word transfer (LODSW), the SI is modified by two. No other Flags are affected by this instruction.

STOS: Store String Byte or String Word

The STOS instruction Stores the AL / AX register contents to a location in the string pointer by ES: DI register pair. The DI is modified accordingly, No Flags are affected by this instruction.

The direction Flag controls the String instruction execution, The source index SI and Destination Index DI are modified after each iteration automatically. If DF=1, then the execution follows auto decrement mode, SI and DI are decremented automatically after each iteration. If DF=0, then the execution follows auto increment mode. In this mode, SI and DI are incremented automatically after each iteration.

7 FLAG MANIPULATION AND A PROCESSOR CONTROL INSTRUCTIONS

These instructions control the functioning of the available hardware inside the processor chip. These instructions are categorized into two types:

1. Flag Manipulation instructions.

2. Machine Control instructions.

Flag Manipulation instructions

The Flag manipulation instructions directly modify some of the Flags of 8086.

i. CLC – Clear Carry Flag.

ii. CMC – Complement Carry Flag.

iii. STC – Set Carry Flag.

iv.CLD – Clear Direction Flag.

v. STD – Set Direction Flag.

vi.CLI – Clear Interrupt Flag.

vii.STI – Set Interrupt Flag.

8 MACHINE CONTROL INSTRUCTIONS

The Machine control instructions control the bus usage and execution

i. WAIT – Wait for Test input pin to go low.

ii. HLT – Halt the process.

iii. NOP – No operation.

iv.ESC – Escape to external device like NDP

v. LOCK – Bus lock instruction prefix.

ASSEMBLER DIRECTIVES:

Assembler directives help the assembler to correctly understand the assembly language programs to prepare the codes. Another type of hint which helps the assembler to assign a particular constant with a label or initialize particular memory locations or labels with constants is called an operator. Rather, the operators perform the arithmetic and logical tasks unlike directives that just direct the assembler to correctly interpret the program to code it appropriately. The following directives are commonly used in the assembly language programming practice using Microsoft Macro Assembler

DB: Define Byte

The DB directive is used to reserve byte or bytes of memory locations in the available memory. While preparing the EXE file, this directive directs the assembler to allocate the specified number of memory bytes to the said data type that may be a constant, variable, string, etc. Another option of this directive also initializes the reserved memory bytes with the ASCII codes of the characters specified as a string. The following examples show how the DB directive is used for different purposes.

Example:

LIST DB 0lH, 02H, 03H, 04H

This statement directs the assembler to reserve four memory locations for a list named LIST and initialize them with the above specified four values.

A microprocessor is a computer processor that performs the tasks of a Central Processing Unit (CPU) on a single integrated circuit (IC). It handles output devices and processes the instructions stored in its memory and provides the output. These processors consist of combinational as well as sequential digital circuits. Moreover, Assembly language is a programing language that helps to program the microprocessors. Overall, Macro and Procedure are two concepts in Microprocessor programming.

The **main difference** between Macro and Procedure is that the **Macro is used for a small number of instructions;** less than ten instructions, but Procedure is used for a large number of instructions; higher than ten instructions.

MACRO VERSUS PROCEDURE

MACRO	PROCEDURE
Sequence of instructions that is written within the macro definition to support modular programming	Set of instructions which can be called repetitively that performs a specific task
Requires more memory	Requires less memory
Does not require CALL and RET instructions	Requires CALL and RET instructions
Machine code is generated each time the macro is called	Machine code generates only once
Parameters are passed as a part of statement which calls the macro	Parameters are passed in registers and memory locations of stack
Macro executes faster than a procedure	Procedure executes slower than a macro
Eliminates the overhead time to call the procedure and to return the program	Requires more overhead time to call the procedure and to return back to the calling procedure
Used for less than 10 instructions	Used for more than 10 instructions

What is Macro

A macro is a set of instructions which has a name, and the programmer can use it anywhere in the program. The main objective of Macros is to achieve modular programming. Furthermore, a macro begins with the %macro directive and ends with the %endmacro directive.

The syntax of Macro is as follows.

%macro macro_name number_of_params

<macro body>

%endmacro

The macro_name helps to identify the macro and the number_of_params refers to the number parameters. Additionally, it is possible to invoke the macro using the macro name with the required parameters. Therefore, if it is necessary to execute the same set of instructions multiple times, the programmer can write those instructions in a macro and use that in his program.

What is Procedure

Procedures are useful to make a large program easier to read, maintain and modify. Usually, a procedure consists of three main sections. Firstly, the procedure name that helps to identify the procedure. Secondly, the statements inside the body, which describes the task to perform. Finally, the return statement that denotes the return statement.

The syntax of Macro is as follows.

```
proc_name:
        procedure body
        …
        RET
```

Furthermore, some functions call a procedure using the CALL instruction. That instruction is as follows.

CALL procedure_name

Finally, after executing the procedure, the control passes to the calling procedure using RET instruction.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF SCIENCE AND HUMANITIES

DEPARTMENT OF PHYSICS

UNIT - IV

Microprocessor and Microcontroller – SPH1313

I/O INTERFACING

Interfacing I/O Devices

- Using I/O devices data can be transferred between the microprocessor and the outside world.
- This can be done in groups of 8 bits using the entire data bus. This is called parallel I/O.
- The other method is serial I/O where one bit is transferred at a time using the SID and SOD pins on the microprocessor.

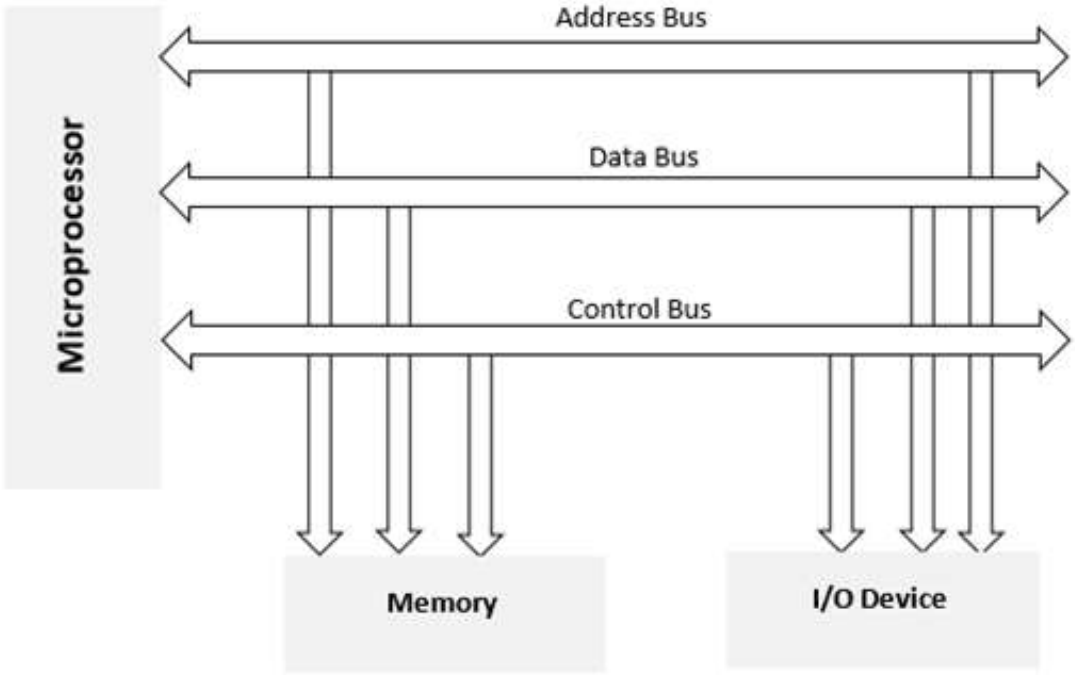


Figure. 1 Block Diagram of Memory and I/O Interfacing

Types of Parallel Interface

- There are two ways to interface 8085 with I/O devices in parallel data transfer mode:

– Memory Mapped IO

– IO Mapped IO

1. I/O mapped I/O:

• It treats them separately from memory.

– I/O devices are assigned a “port number” within the 8-bit address range of 00H to FFH.

– The user in this case would access these devices using the IN and OUT instructions only.

8086 has special instructions IN and OUT to transfer data through the input/output ports in I/O mapped I/O system. The IN instruction copies data from a port to the Accumulator. If an 8-bit port is read data will go to AL and if 16-bit port is read the data will go to AX. The OUT instruction copies a byte from AL or a word from AX to the specified port. The M/IO signal is always low when 8086 is executing these instructions. In this address of I/O device is 8-bit or 16-bit. It is 8-bit for Direct addressing and 16-bit for Indirect addressing.

2. Memory mapped I/O

• It considers them like any other memory location.

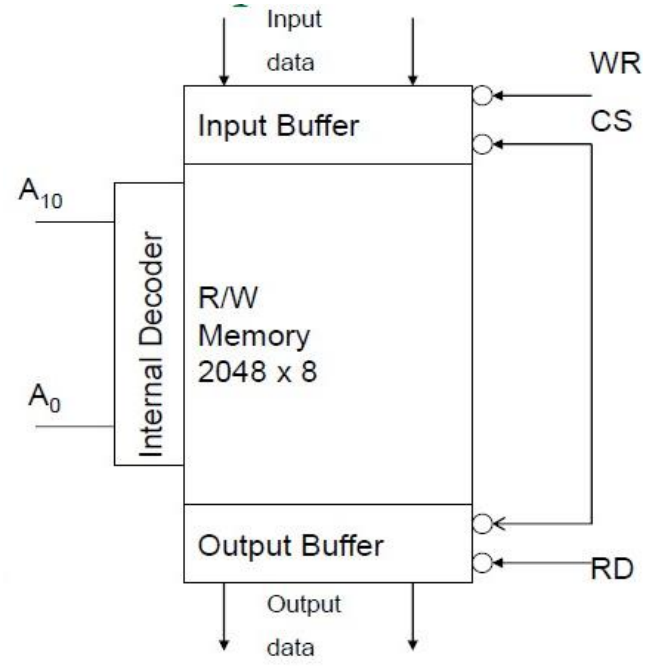
– They are assigned a 16-bit address within the address range of the 8085.

– The exchange of data with these devices follows the transfer of data with memory. The user uses the same instructions used for memory.

In this type of I/O interfacing, the 8086 uses 20 address lines to identify an I/O device. The I/O device is connected as if it is a memory device. The 8086 uses same control signals and instructions to access I/O as those of memory, here RD and WR signals are activated indicating memory bus cycle.

Memory Structure and its requirements

R/W memories consist of an array of registers, in which each register has unique address The sixe of the memory is N*M as shown in figure 2, where N is the number of registers and M is the word length in bits.



Logic Diagram for RAM

Example

- If memory is having 12 address lines and 8 data lines, then Number of registers/ memory locations (capacity) = 2N= 212
- = 4096
- Word length = M bit
- = 8 bit

Example: If memory has 8192 memory locations, then it has 13 address lines.

Table summarizes capacity with address in table 1

Table 1. Memory capacity with address lines required

Memory Capacity	Address lines required
1k = 1024 memory locations	10
2k = 2048 memory locations	11
4k = 4096 memory locations	12
8k = 8192 memory locations	13
16k = 16384 memory locations	14
32k = 32768 memory locations	15
64k = 65536 memory locations	16

IO mapped IO V/s Memory Mapped IO

Memory Mapped I/O

- IO is treated as memory.
- 16-bit addressing.
- More Decoder Hardware.
- Can address 216=64k locations.
- Less memory is available.

I/O Mapped IO

- IO is treated IO.
- 8- bit addressing.
- Less Decoder Hardware.
- Can address 28=256 locations.
- Whole memory address space is available.

IO mapped IO V/s Memory Mapped IO

- Memory Mapped IO
- Memory Instructions are used.
- Memory control signals are used.
- Arithmetic and logic operations can be performed on data.
- Data transfer b/w register and IO.

- IO Mapped IO
- Special Instructions are used like IN, OUT.
- Special control signals are used.
- Arithmetic and logic operations can not be performed on data.
- Data transfer b/w accumulator and IO.

The interfacing of output devices

- Output devices are usually slow.
- Also, the output is usually expected to continue appearing on the output device for a long period of time.

- Given that the data will only be present on the data lines for a very short period (microseconds), it has to be latched externally.

The interfacing of output devices

- To do this the external latch should be enabled

when the port's address is present on the address bus, the IO/M signal is set high and WR is set low.

- The resulting signal would be active when the output device is being accessed by the microprocessor.
- Decoding the address bus (for memory-mapped devices) follows the same techniques discussed in interfacing memory.
- The objective of interfacing I/O peripherals:

– is to obtain information or results from process.

– to store, process or display.

- The instructions IN and OUT perform this operation.
- The following examples shows the process of instruction:

2050 D3 OUT 01H

2051 01 IN 02H

Parallel communication interface

The 8255A is a general purpose programmable I/O device designed to transfer the data from I/O to interrupt I/O under certain conditions as required.

It can be used with almost any microprocessor.

It consists of three 8-bit bidirectional I/O ports (24 I/O lines)

It is flexible versatile, economic but complex.

Ports of 8255A

8255A has three ports, i.e., PORT A, PORT B, and PORT C.

Port A contains one 8-bit output latch/buffer and one 8-bit input buffer.

Port B is similar to PORT A.

Port C can be split into two parts, i.e. PORT C lower (PC0-PC3) and PORT C upper (PC7-PC4) by the control word.

8255A Operating modes

Bit set reset (BSR) mode

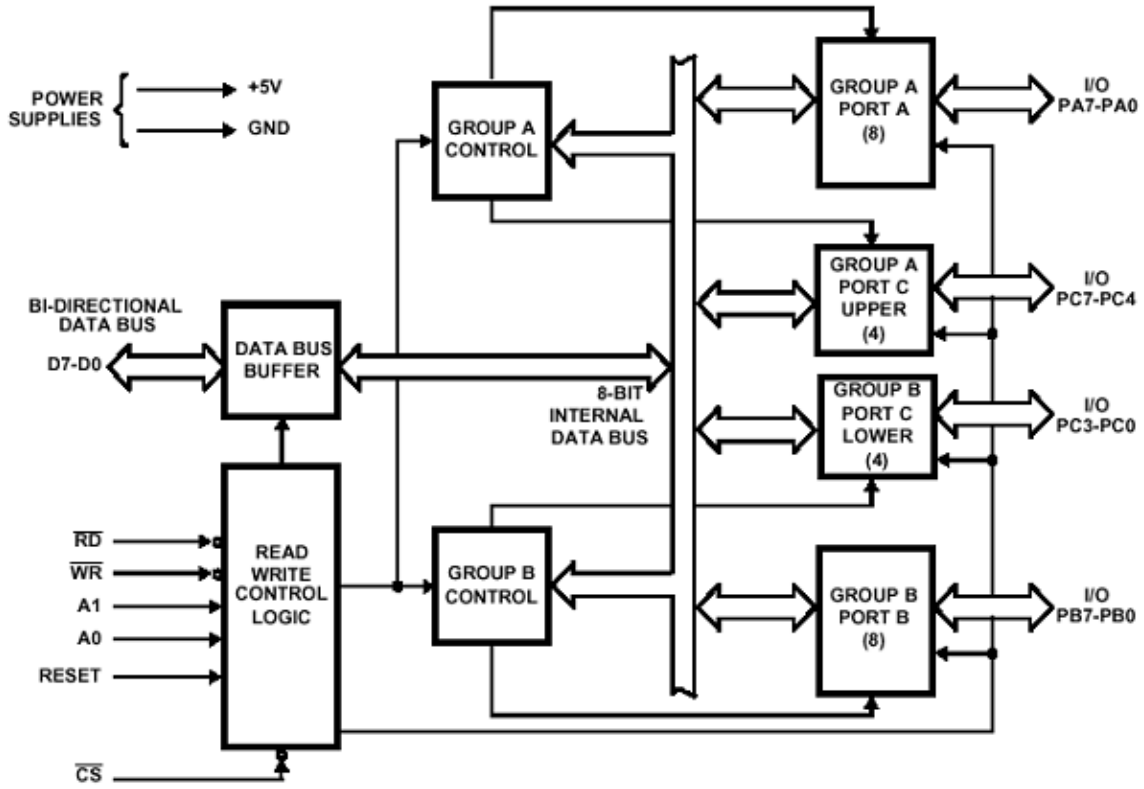
Input-Output mode [Mod 0, Mod 1, Mod 2]

Control signals of 8255 that are used for interfacing with a microprocessor

RD*	It is an active low input pin for 8255. It is connected to RD* output of 8085. The 8085 activates the RD* input of 8255 when it wants to read the data present in a port of 8255.
WR*	It is an active low input pin for 8255. It is connected to WR* output of 8085. The 8085 activates the WR* input of 8255 when it wants to write data to a port of 8255.
CS, A1, A0	CS is an active low input pin for 8255. A1, A0 are address-input pins. They select one of the ports inside 8255 for communication with the microprocessor.
Reset	It is an active high input pin. It is connected to ResetOut output of 8085. It is used to reset the 8255. After a reset of 8255, all the three ports of 8255 work as input ports in mode 0, which is the simplest mode of operation. Operational modes of Ports are described later.

CS	A1	A0	SELECTION	ADDRESS
0	0	0	PORT A	80 H
0	0	1	PORT B	81 H
0	1	0	PORT C	82 H
0	1	1	Control	83 H

			Register	
1	X	X	No Seletion	X



BLOCK DIAGRAM OF 8255 PPI

It consists of 40 pins and operates in +5V regulated power supply. Port C is further divided into two 4-bit ports i.e. port C lower and port C upper and port C can work in either BSR (bit set rest) mode or in mode 0 of input-output mode of 8255. Port B can work in either mode or in mode 1 of input-output mode. Port A can work either in mode 0, mode 1 or mode 2 of input-output mode

It has two control groups, control group A and control group B. Control group A consist of port A and port C upper. Control group B consists of port C lower and port B.

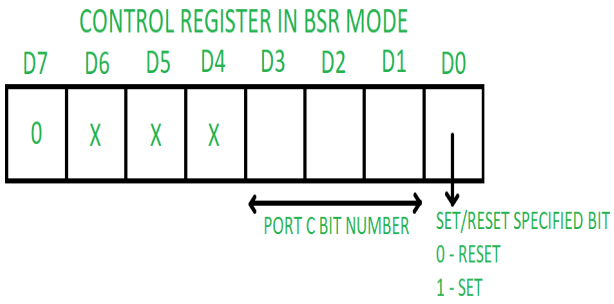
Depending upon the value if CS', A1 and A0 we can select different ports in different modes as input-output function or BSR. This is done by writing a suitable word in control register (control word D0-D7).

CS'	A1	A0	SELECTION	ADDRESS
0	0	0	PORT A	80 H
0	0	1	PORT B	81 H
0	1	0	PORT C	82 H
0	1	1	Control Register	83 H
1	X	X	No Seletion	X

Operating modes –

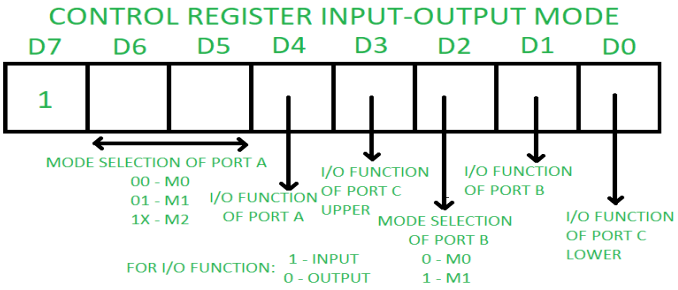
1. **Bit set reset (BSR) mode –**

   If MSB of control word (D7) is 0, PPI works in BSR mode. In this mode only port C bits are used for set or reset.



2. **Input-Output mode –**

   If MSB of control word (D7) is 1, PPI works in input-output mode. This is further divided into three modes:



- **Mode 0** –In this mode all the three ports (port A, B, C) can work as simple input function or simple output function. In this mode there is no interrupt handling capacity.
- **Mode 1** – Handshake I/O mode or strobbed I/O mode. In this mode either port A or port B can work as simple input port or simple output port, and port C bits are used for handshake signals before actual data transmission. It has interrupt handling capacity and input and output are latched.

  Example: A CPU wants to transfer data to a printer. In this case since speed of processor is very fast as compared to relatively slow printer, so before actual data transfer it will send handshake signals to the printer for synchronization of the speed of the CPU and the peripherals.

- **Mode 2 –** Bi-directional data bus mode. In this mode only port A works, and port B can work either in mode 0 or mode 1. 6 bits port C are used as handshake signals. It also has interrupt handling capacity.

8251 Programmable Serial communication interface

(USART – Universal synchronous asynchronous receiver transmitter)

8251 universal synchronous asynchronous receiver transmitter (USART) acts as a mediator between microprocessor and peripheral to transmit serial data into parallel form and vice versa.

1. It takes data serially from peripheral (outside devices) and converts into parallel data.
2. After converting the data into parallel form, it transmits it to the CPU.
3. Similarly, it receives parallel data from microprocessor and converts it into serial form.
4. After converting data into serial form, it transmits it to outside device (peripheral).

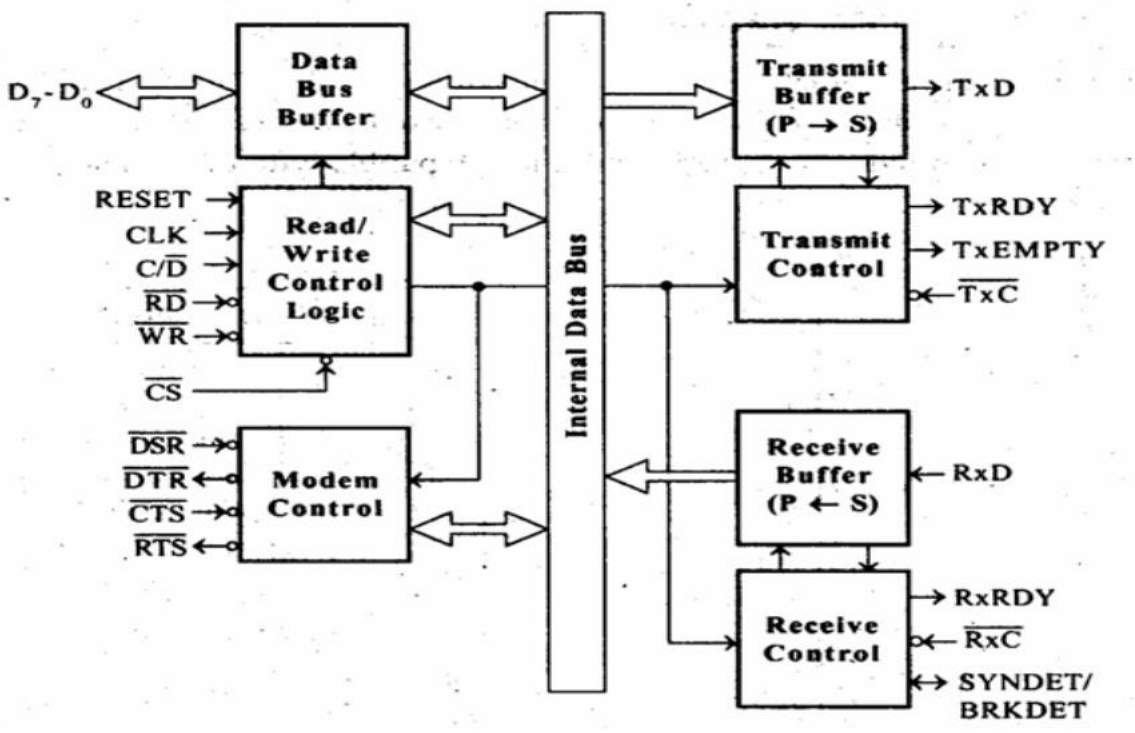


Figure . block diagram of 8251

It contains the following blocks:

1. **Data bus buffer –**

   This block helps in interfacing the internal data bus of 8251 to the system data bus. The data transmission is possible between 8251 and CPU by the data bus buffer block.

2. **Read/Write control logic –**

   It is a control block for overall device. It controls the overall working by selecting the operation to be done. The operation selection depends upon input signals as:

$\overline{CS}$	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	Operation
1	X	X	X	Invalid
0	0	0	1	data CPU< ----- 8251
0	0	1	0	data CPU ----- > 8251
0	1	0	1	Status word CPU < ----------8251
0	1	1	0	Control word CPU---------- > 8251

   In this way, this unit selects one of the three registers- data buffer register, control register, status register.

3. **Modem control (modulator/demodulator) –**

   A device converts analog signals to digital signals and vice-versa and helps the computers to communicate over telephone lines or cable wires. The following are active-low pins of Modem.

   - **DSR:** Data Set Ready signal is an input signal.
   - **DTR:** Data terminal Ready is an output signal.
   - **CTS:** It is an input signal which controls the data transmit circuit.

     RTS: It is an output signal which is used to set the status RTS.

4. **Transmit buffer –**

   This block is used for parallel to serial converter that receives a parallel byte for conversion into serial signal and further transmission onto the common channel.

- **TXD:** It is an output signal, if its value is one, means transmitter will transmit the data.

5. **Transmit control –**

   This block is used to control the data transmission with the help of following pins:

   - **TXRDY:** It means transmitter is ready to transmit data character.
   - **TXEMPTY:** An output signal which indicates that TXEMPTY pin has transmitted all the data characters and transmitter is empty now.
   - **TXC:** An active-low input pin which controls the data transmission rate of transmitted data.

6. **Receive buffer –**

   This block acts as a buffer for the received data.

   - **RXD:** An input signal which receives the data.

7. **Receive control –**

   This block controls the receiving data.

   - **RXRDY:** An input signal indicates that it is ready to receive the data.
   - **RXC:** An active-low input signal which controls the data transmission rate of received data.
   - **SYNDET/BD:** An input or output terminal. External synchronous mode-input terminal and asynchronous mode-output terminal.

Mode of Operation of 8251

Once the 8251 is programmed as, required, the TXRDY is raised high to signal the CPU that 8251 is ready to receive data byte from it that is to be converted in to serial format and transmitted. This automatically goes low when the CPU writes the data in to 8251. • Two functional types : • Mode instruction control word • Command Instruction control word. • Two modes of operation:

• Asynchronous mode

• Synchronous mode

Asynchronous Mode (Transmission)

• When a data character is sent to 8251A by the CPU, it adds start bits prior to the serial data bits, followed by optional parity bit and stop bits using the asynchronous mode instruction control word format.

• This sequence is then transmitted using TXD output pin on the falling edge of TXC.

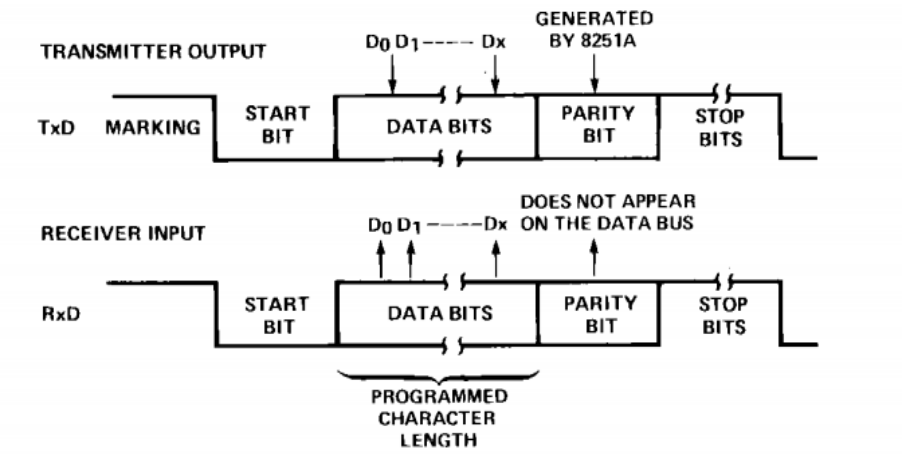


Fig. Data format for Asynchronous mode

Asynchronous Mode (Receive)

• A falling edge on RXD input line marks a start bit.

• The receiver requires only one stop bit to mark end of the data bit string, regardless of the stop bit programmed at the transmitting end. The 8-bit character is then loaded into the parallel I/O buffer of 8251.

• RXRDY pin is raised high to indicate to the CPU that a character is ready for it. • If the previous character has not been read by the CPU, the new character replaces it, and the overrun flag is set indicating that the previous character is lost.

Synchronous Mode (Transmission)

• The TXD output is high until the CPU sends a character to 8251 which usually is a SYNC character.

• When line goes low, the first character is serially transmitted out. • Characters are shifted out on the falling edge .

• Data is shifted out at the same rate as , over TXD output line.

• If the CPU buffer becomes empty, the SYNC character or characters are inserted in the data stream over TXD output.

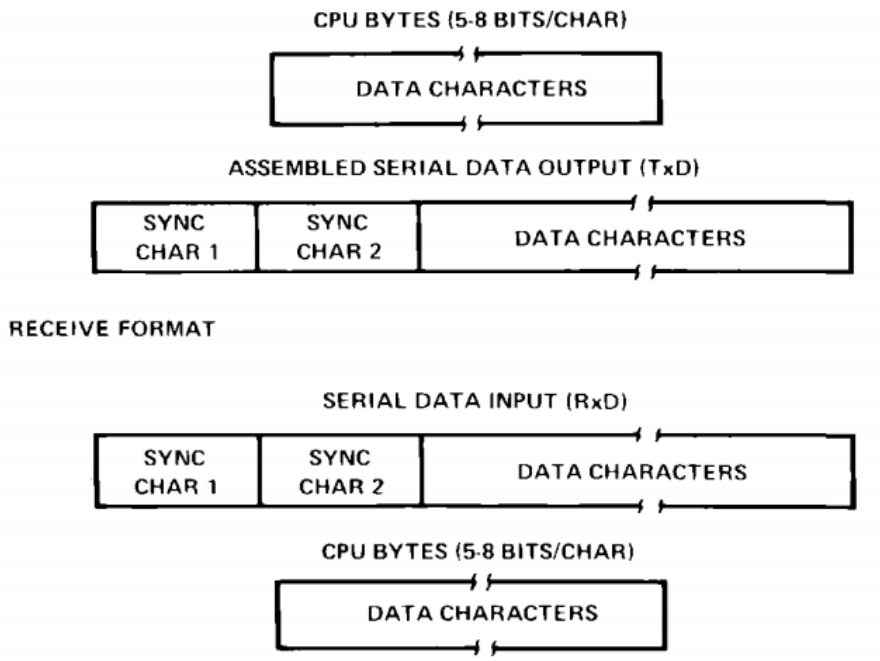


Fig. Data format for Synchronous mode

Synchronous Mode (Receiver)

• In this mode, the character synchronization can be achieved internally or externally. • The data on RXD pin is sampled on rising edge .

• The content of the receiver buffer is compared with the first SYNC character at every edge until it matches.

• If 8251 is programmed for two SYNC characters, the subsequent received character is also checked.

• When the characters match, the hunting stops.

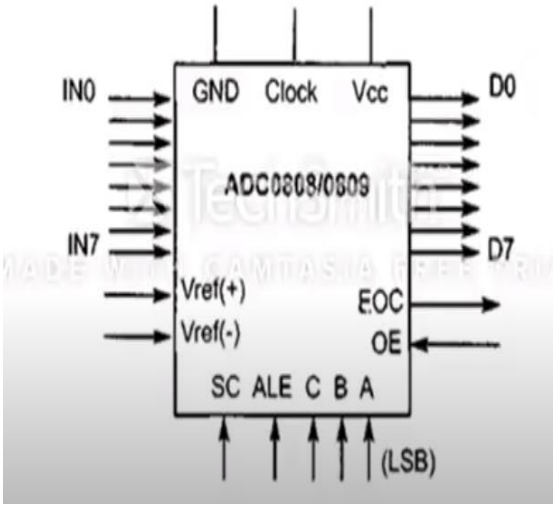
• The SYNDET pin set high and is reset automatically by a status read operation. • In the external SYNC mode, the synchronization is achieved by applying a high level on the SYNDET input pin that forces 8251 out of HUNT mode.

• The high level can be removed after one cycle.

• The parity and overrun error both are checked in the same way as in asynchronous mode.

A/D Interface

- ADC 0809 is an 8 channel 8 bit ADC i.e. it divides the voltage applied at Vref+ & Vref- into 28 i.e. 256 steps.
- Conversion delay is 100 micro seconds
- Operating speed 640 KHz(clk freq)
- 28 pin dip
- Uses successive appx method



Conversion delay:

Time taken by ADC from active edge of SOC to active edge of EOC pulse (100µs)

Algorithm

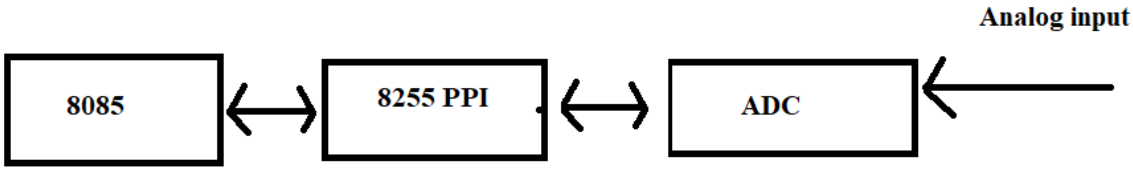
Ensures the stability of analog signal applied to the ADC

Issues SOC pulse to the µP

Read EOC to mark the conversion

Read digital output of ADC as equivalent to ADC

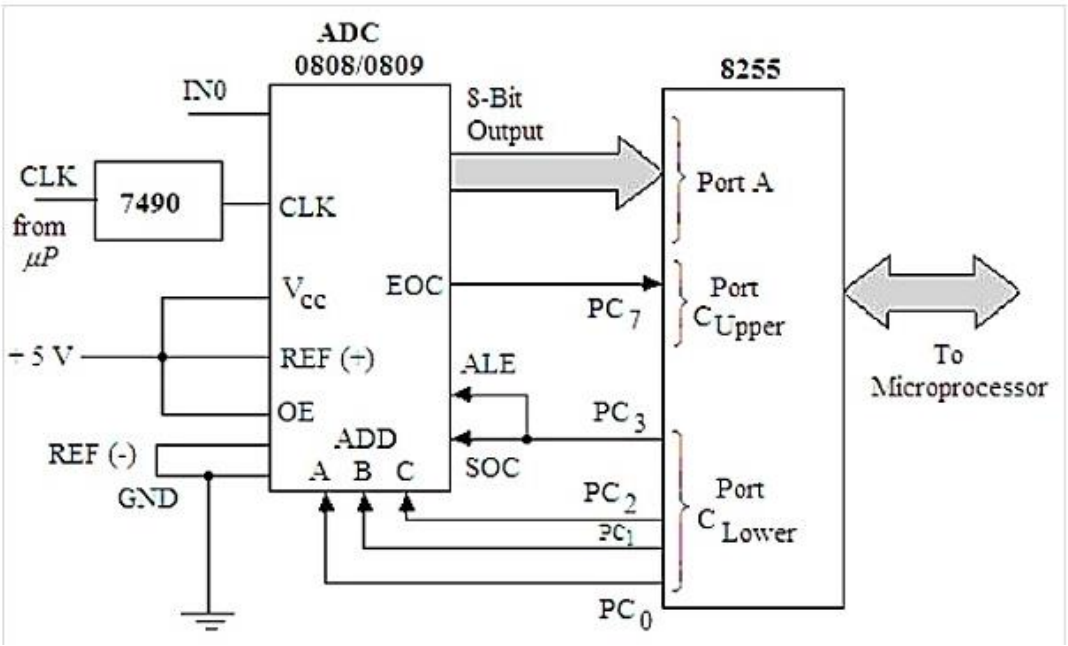
- It internally contains 3:8 analog multiplexer
- Out of these 8 i/ps only one can be selected for conversion using 3 i/p address lines
- They do not contain any internal sample and hold circuit



Selected Analog Channel	C	B	A
IN0	0	0	0
IN1	0	0	1
IN2	0	1	0
IN3	0	1	1
IN4	1	0	0
IN5	1	0	1
IN6	1	1	0
IN7	1	1	1

Interfacing ADC with 8085 Microprocessor

To interface the ADC with 8085, we need 8255 Programmable Peripheral Interface chip with it. Let us see the circuit diagram of connecting 8085, 8255 and the ADC converter.



- ADC 0809 is an 8 channel, 8 bit ADC. It can convert an analog voltage input into an 8 bit digital data output.
- To select an input out of 8 options, there are three select lines (C, B and A). We put a channel number on these lines (0…7) and latch it using ALE. SOC signal is given to indicate start of conversion.

- The channel voltage is internally sampled and held into a capacitor. Conversion takes place internally using “Successive Approximations Algorithm”.
- Reference voltage for conversion is provided using +Vref and –Vref. The clock supply needed for conversion is given through CLK (typically ~ 1MHz).
- The end of conversion is indicated by the ADC using EOC signal. Now we give the OE signal enabling 8-bit data output from the ADC to 8255.
- This data from 8255 is now transferred to the microprocessor. The process is repeated for subsequent channels, by changing the channel number. ADCs have a vast use in the modern electronic world for Data Acquisition Systems. They can be used for temperature sensing, voice recording, speed sensing etc.

- The PortA of 8255 chip is used as the input port.
- The PC_7 pin of Port C_{upper} is connected to the End of Conversion (EOC) Pin of the analog to digital converter. This port is also used as input port. The C_{lower} port is used as output port.
- The $PC_{2\text{-}0}$ lines are connected to three address pins of this chip to select input channels.
- The PC_3 pin is connected to the Start of Conversion (SOC) pin and ALE pin of ADC 0809.

Program:

- MVI A, 98H ;Initlization 8255; Set Port A and Cupper as input, CLower as output
- OUT 83H ; Write control word 8255 to control Word register
- XRA A ; Clear the accumulator
- OUT 82H ; Send the content of Acc to Port Clower to select IN0
- MVI A, 08H ; Load the accumulator with 08H
- OUT 82H ; ALE and SOC will be logic ‘1'

- XRA A ; Clear the accumulator
- OUT 82H ; ALE and SOC will be low.
- READ: IN 82H ; Read from EOC (PC7) ; check for EOC
- RAL ; Rotate left to check C7 is 1.
- JNC READ ; If C7 is not 1, go to READ
- IN 80H ; Read digital output of ADC
- STA 8000H ; Save result at 8000H
- HLT ; Stop the program

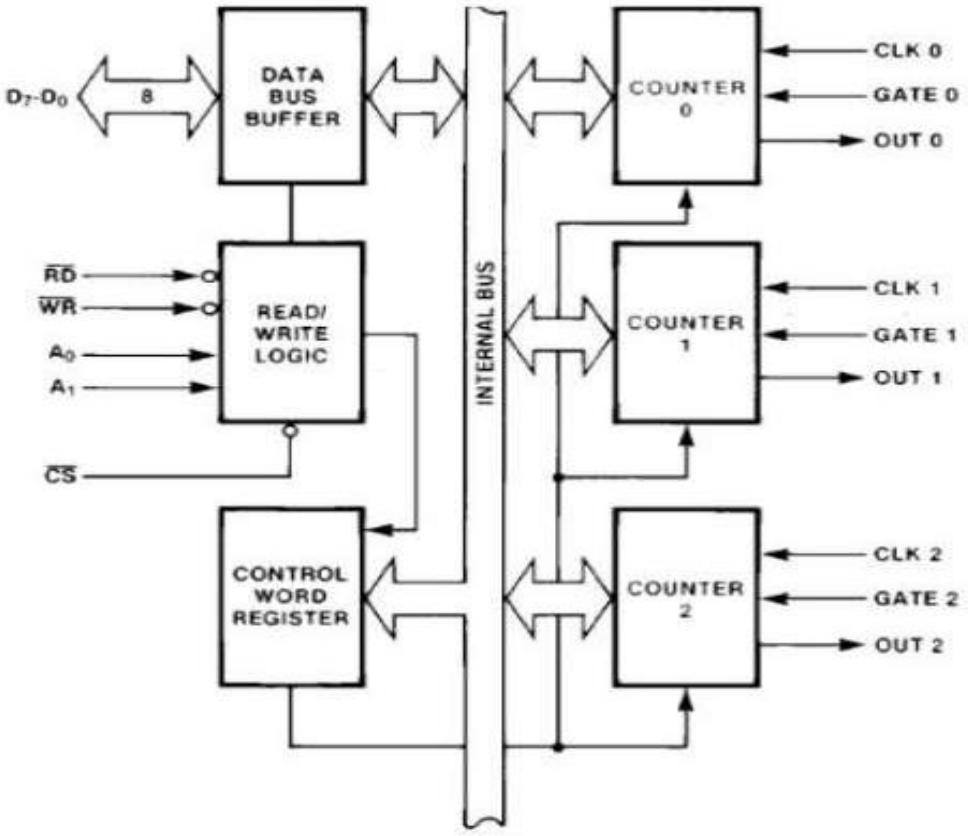
8254 programmable interval timer

8254 is a device designed to solve the timing control problems in a microprocessor

Features of 8254

- It has three independent 16-bit down counters.
- It can handle inputs from DC to 10 MHz.
- These three counters can be programmed for either binary or BCD count.
- It is compatible with almost all microprocessors.
- It can operate in 6 independent modes
- 8254 has a powerful command called READ BACK command, which allows the user to check the count value, the programmed mode- the current mode, and the current status of the counter.
- Designed for microprocessors to perform timing and counting functions using three 16-bit registers.
- Each counter has 2 input pins, i.e. Clock & Gate, and pin for “OUT” output.
- To operate a counter, a 16-bit count is loaded in its register.
- On command, it begins to decrement the count until it reaches 0, then it generates a pulse that can be used to interrupt the CPU.

Architecture of 8254



- It consist of four major blocks
- Data Bus Buffer
- READ/WRITE Logic
- Control Word Register
- Counters

Data Bus Buffer

- t is a tri-state, bi-directional, 8-bit buffer, which is used to interface the 8254 to the system data bus.
- It controls the transmission of data between Processor and 8354 through IN/OUT instructions
- It has three basic functions −
- Select the programming modes of 8254.(Control word)
- Load the count registers.
- Load the count values.

READ/WRITE Logic

It includes 5 signals, i.e. RD, WR, CS, and the address lines A_0 & A_1.
In the peripheral I/O mode, the RD and WR signals are connected to IOR and IOW, respectively.
In the memory mapped I/O mode, these are connected to MEMR and MEMW.
Address lines A_0 & A_1 of the CPU are connected to lines A_0 and A_1 of the 8254, and CS is tied to a decoded address.
The control word register and counters are selected according to the signals on lines A_0 & A_1.

A_1	A_0	Result
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control Word Register
X	X	No Selection

Counters

- Each counter consists of a single, 16 bit-down counter, which can be operated in either binary or BCD.
- Its input and output is configured by the selection of modes stored in the control word register
- The programmer can read the contents of any of the three counters without disturbing the actual count in process.

Control Word Register

- This register is accessed when lines A_0 & A_1 are at logic 1.
- It is used to write a command word, which specifies the counter to be used, its mode, and either a read or write operation

Control Word Format

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC—Select Counter

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (see Read Operations)

M—Mode

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

RW—Read/Write

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only
1	0	Read/Write most significant byte only
1	1	Read/Write least significant byte first, then most significant byte

BCD

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

Operational Modes

Mode 0 ─ Interrupt on Terminal Count

Mode 1 – Programmable One Shot

Mode 2 – Rate Generator

Mode 3 – Square Wave Generator

Mode 4 − Software Triggered Mode

Mode 5 – Hardware Triggered Mode

To initialize counter

Write the control word in control register, write the count value in count register

8279 - PROGRAMMABLE KEYBOARD

Features

- Used for interfacing Keyboard /display devices to the microprocessor based system.
- Simultaneous Display and Keyboard operation
- It provides 3 operating input modes
    - ✓ Scanned Keyboard mode
    - ✓ Scanned Sensor mode
    - ✓ Strobed Input mode
- It has inbuilt debounce key
- 8 character Keyboard FIFO
- 16 character display; 16 byte RAM
    - ✓ Left entry (Type writer mode)
    - ✓ Right Entry (calculator mode)

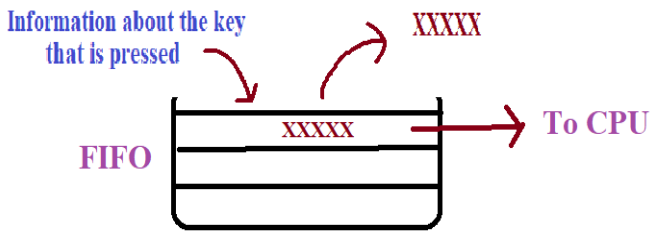
The ways the Keyboard can be interfaced with the processor

Interrupt Mode	Polled mode
The processor is requested service only if any key is pressed, otherwise the CPU will continue with its main task.	The CPU periodically reads an internal flag of 8279, to check whether any key is pressed or not.

How does Keyboard work

Keyboard consist of maximum 64 keys which are interfaced with microprocessor using keycodes.

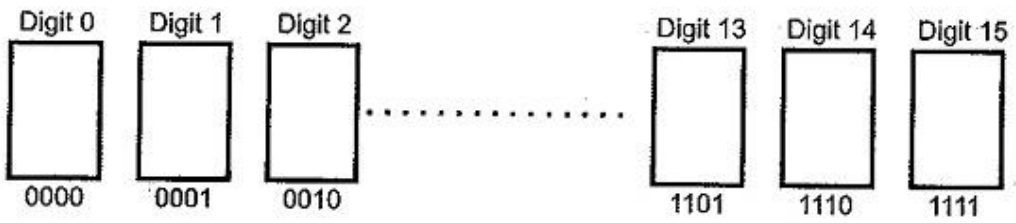
- These key-codes are de-bounced and stored in an 8-byte FIFO RAM, which can be accessed by the CPU.
- If more than 8 characters are entered in the FIFO, then it means more than eight keys are pressed at a time. This is when the over run status is set.
- If a FIFO contains a valid key entry, then the CPU is interrupted in an interrupt mode else the CPU checks the status in polling mode to read the entry.
- Once the CPU reads a key entry, then FIFO is updated, and the key entry is pushed out of the FIFO to generate space for new entries



OUTPUT MODES

Left Entry (type writer type)

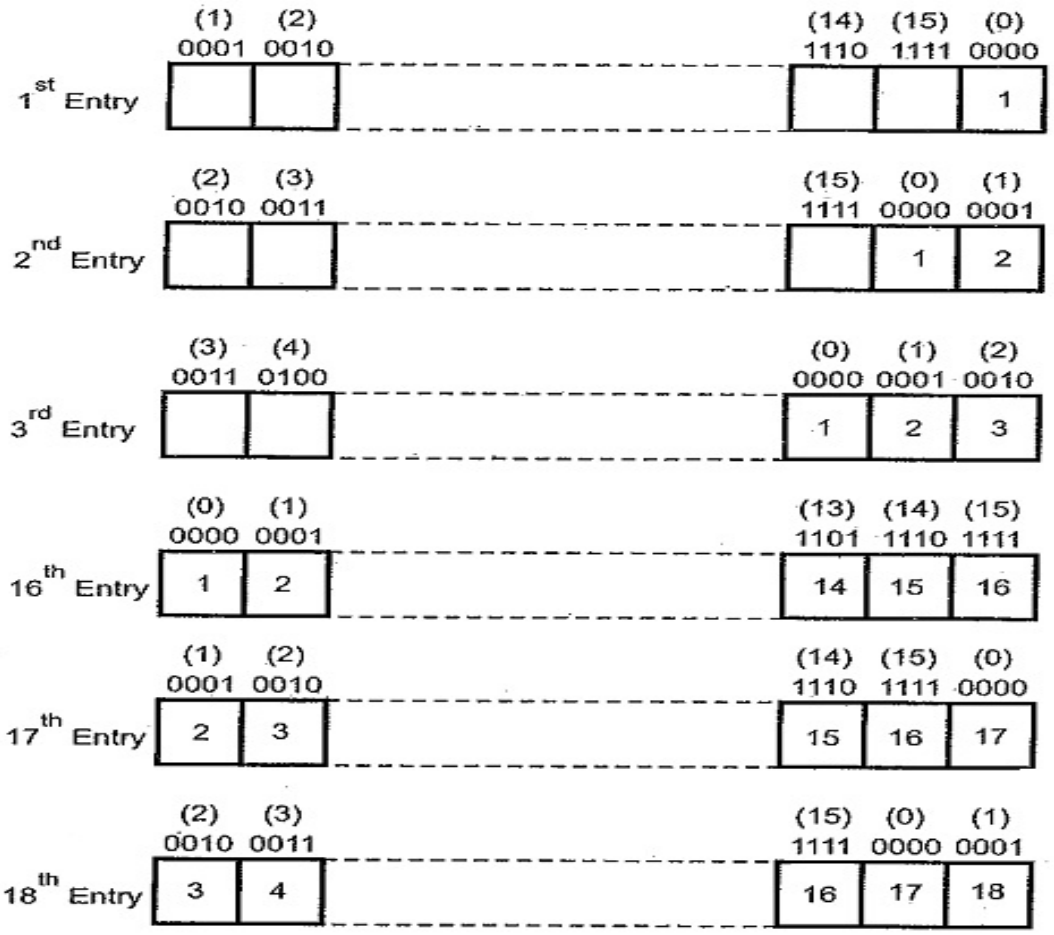
In the left entry mode, Operating Modes of 8279 displays characters from left to right in the multiplexed displays like a typewriter. In this, each display position is directly corresponds to a byte (or nibble) in the display RAM. Address 0 in the RAM is the left-most display character and address 15 (or address 7 in 8 character display) is the right most display character



(b) 16-character display left entry mode

Right Entry

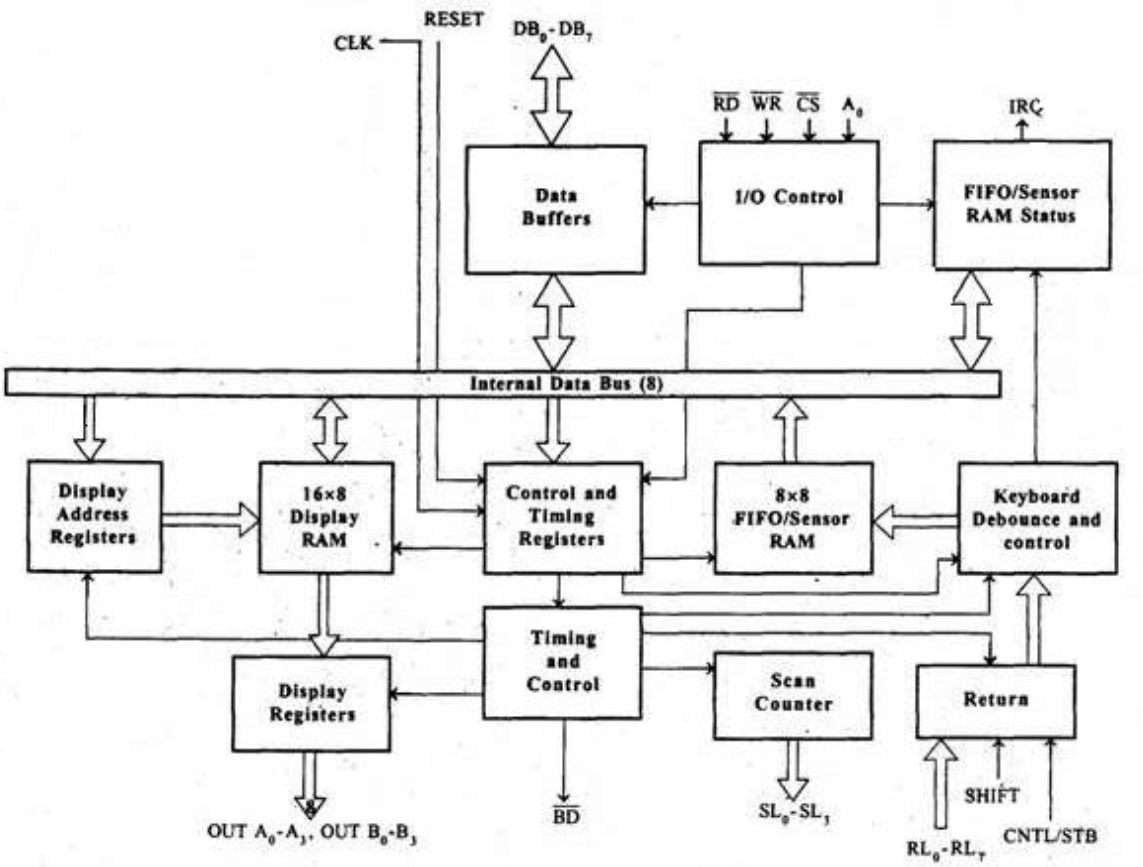
In the right entry mode, Operating Modes of 8279 displays characters from right to left in the multiplexed display like a calculator. The first entry is displayed on the right most display. The next entry is also displayed on the right most display after the display is shifted left one character



Architecture of 8279

8279 consist of four major Sections

- CPU interface and Control Section
- Scan Section
- Keyboard Section
- Display Section



CPU Interface Section

Data bus buffer

- This unit takes care of data transfer between 8279 and the processor
- 8 bidirectional data lines DB0 – DB7

I/O control

- It requires two internal address A0 = 0 for selecting data buffer and A0 = 1 for selecting control register.
- Control signals
- WR, RD, CS

Timing and control Register

Stores the keyboard modes and other operating condition programmed in processor

Timing and control Logic

9279 requires an internal clock frequency of 100 KHz

Scan Section

- Scan counter and 4 scan lines SL0 – SL3
- Has two modes, Ecoded mode, Decoded mode

- Encoded scan mode, the o/p of the scan lines will be in binary count, uses all 16 lines(Display mode)
- In decoded mode, the o/p of the scan lines will be similar to a 2- to -4 decoder, decodes 2 LSB bits.

Keyboard Section

- Return lines RL0 – RL7 that can be used to form the columns of a keyboard matrix
- Two additional i/p
  - SHIFT
  - CONTROL/STROBE
- Modes
  - 2 Key lockout
  - N- key roll over
- 8*8 FIFO Sensor RAM
- FIFO RAM – where the keycode of every pressed key is entered as per their sequence
- FIFO RAM can store 8 keycodes

2 key lockout mode

- whenever keys are pressed on a keyboard only one key is recognised and recorded being a debounce feature. Thus, even if multiple keys are pressed simultaneously only one operation will be recorded.
- In N key rollover, simultaneous keys can also be recorded at once and their codes are stored in the first in first out RAM. Thus, not need to wait for a single key at a time process.

Return Buffer

- Contain return lines RL0 – RL7, SHIFT, STB/CONTROL
- In STB mode content of return lines are returned to FIFO RAM
- FIFO sensor and RAM status generates an interrupt signal when there is an entry in FIFO
- Scan keyboard mode – 64 keys
- Scan senor – condition for 64 switches
- Keyboard debounce and control unit-
- Enabled when Key board mode is selected

- First scans the key closure row wise, if found then the unit debounce the key entry.

Display Section

- 8 output lines divided into 2 groups
    - OUT A0 – A3 OUT B0 – B3
- This unit consists of display address registers which handles the addresses of the currently read word /written by the CPU to/from the display RAM.

Programmable Interrupt Controller (8259)

- 8259 microprocessor is defined as **Programmable Interrupt Controller (PIC)** microprocessor.
- There are 5 hardware interrupts and 2 hardware interrupts in 8085 and 8086 respectively.
- By connecting 8259 with CPU, we can increase the interrupt handling capability. 8259 combines the multi interrupt input sources into a single interrupt output.
- Interfacing of single PIC provides 8 interrupts inputs from IR0-IR7.
- For example, Interfacing of 8085 and 8259 increases the interrupt handling capability of 8085 microprocessor from 5 to 8 interrupt levels.
- This chip is designed for 8085 and 8086.
- It can be programmed either in edge triggered, or in level triggered mode
- We can mask individual bits of Interrupt Request Register.
- By cascading 8259 chips, we can increase interrupts up to 64 interrupt lines
- Clock cycle is not needed.

Architecture of 8259

8259 can be divided into following blocks namely

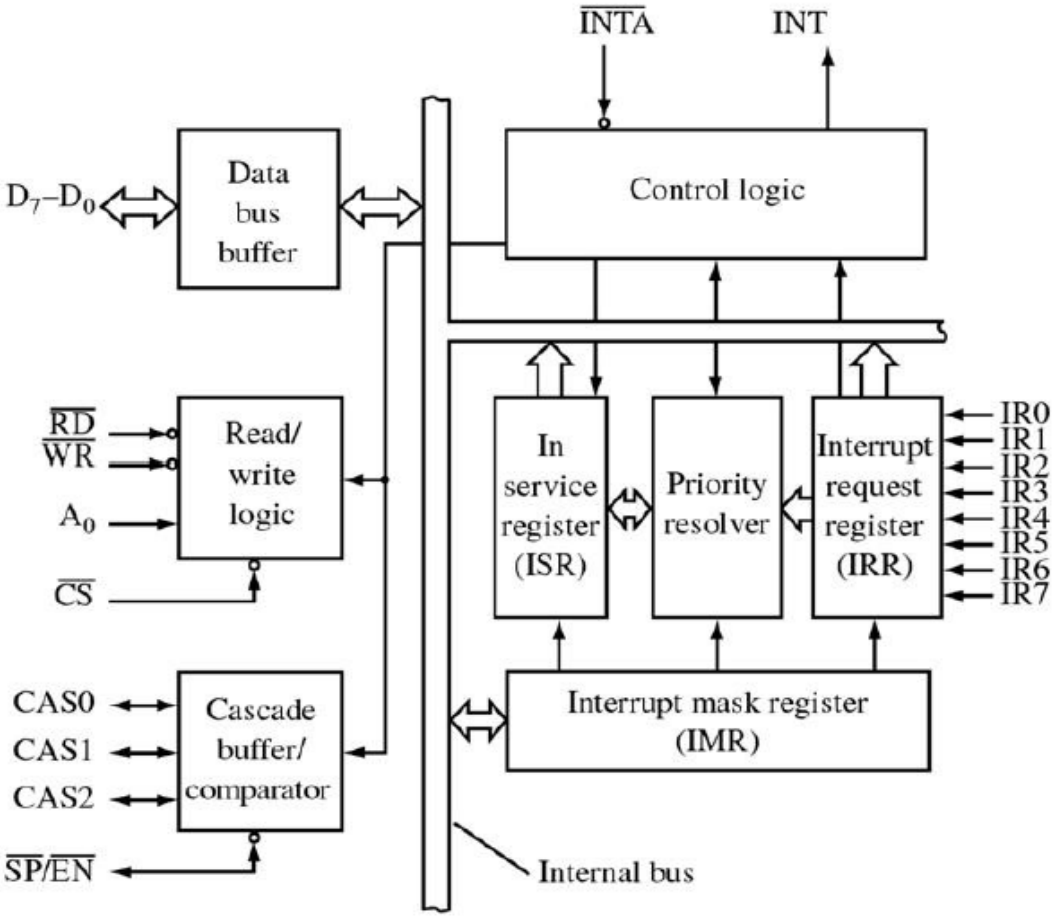


Figure 2. block diagram of 8259

Block	Description
Data Bus Buffer	This block is used to communicate between 8259 and 8085/8086 by acting as buffer. It takes the control word from 8085/8086 and send it to the 8259. It transfers the opcode of the selected interrupts and address of ISR to the other connected microprocessor. It can send maximum 8-bit at a time.

Block	Description
R/W Control Logic	This block works when the value of pin CS is 0. This block is used to flow the data depending upon the inputs of RD and WR. These are active low pins for read and write.
Control Logic	It controls the functionality of each block. It has pin called INTR. This is connected to other microprocessors for taking the interrupt request. The INT pin is used to give the output. If 8259 is enabled, and also the interrupt flags of other microprocessors are high then this causes the value of the output INT pin high, and in this way this chip can responds requests made by other microprocessors.
Interrupt Request Register	It stores all interrupt level that are requesting for interrupt service.
Interrupt Service Register	It stores interrupt level that are currently being execute.
Interrupt Mask Register	It stores interrupt level that will be masked, by storing the masking bits of interrupt level.
Priority Resolver	It checks all three registers, and set the priority of the interrupts. Interrupt with the highest priority is set in the ISR register. It also reset the interrupt level which is already been serviced in the IRR.

Block	Description
Cascade Buffer	To increase number of interrupt pin, we can cascade more number of pins, by using cascade buffer. When we are going to increase the interrupt capability, CSA lines are used to control multiple interrupts.

DMA CONTROLLER 8257

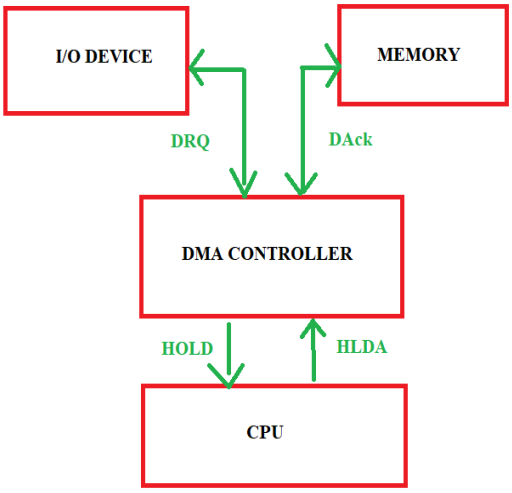
8257 DMA stands for 4-channel Direct Memory Access. It is specially designed by Intel for data transfer at the highest speed. Its initial function is to generate a peripheral request which allows the device to transfer the data directly to/from memory without any interference of the CPU.

Features of 8257

Here is a list of some of the prominent features of 8257 −

- It has four channels which can be exhibited over four I/O devices.
- Each channel has 16-bit address and 14-bit counter.
- Data transfer of each channel can be taken up to 64kb.
- Each channel can be programmed independently.
- Each channel can perform certain specific actions i.e, read transfer, write transfer and verify transfer operations.
- It produces MARK signal to the peripheral device that 128 bytes have been transferred.
- It requires a single phase clock.
- Its frequency ranges from 250Hz to 3MHz.
- It performs operations in 2 modes, i.e., Master mode and Slave mode.

Direct Memory Access (DMA)



- During **any given bus cycle, one of the system components connected to the system bus is given control of the bus.** This component is said to be the **master** during that cycle and the component it is communicating with is said to be the**slave.**
- The **CPU with its bus control logic is normally the master,** but other specially designed components can gain control of the bus by sending a bus request to theCPU.
- After the current bus cycle is completed the CPU will return a bus grant signal and the component sending the request will become themaster.
- Taking control of the bus for a bus cycle is called cyclestealing.
- The DMA data transfer is initiated only after receiving **HLDA signal from theCPU.**
- The 8257, on behalf of the devices, requests the CPU for bus access using local bus request input i.e**. HOLD in minimum mode.**

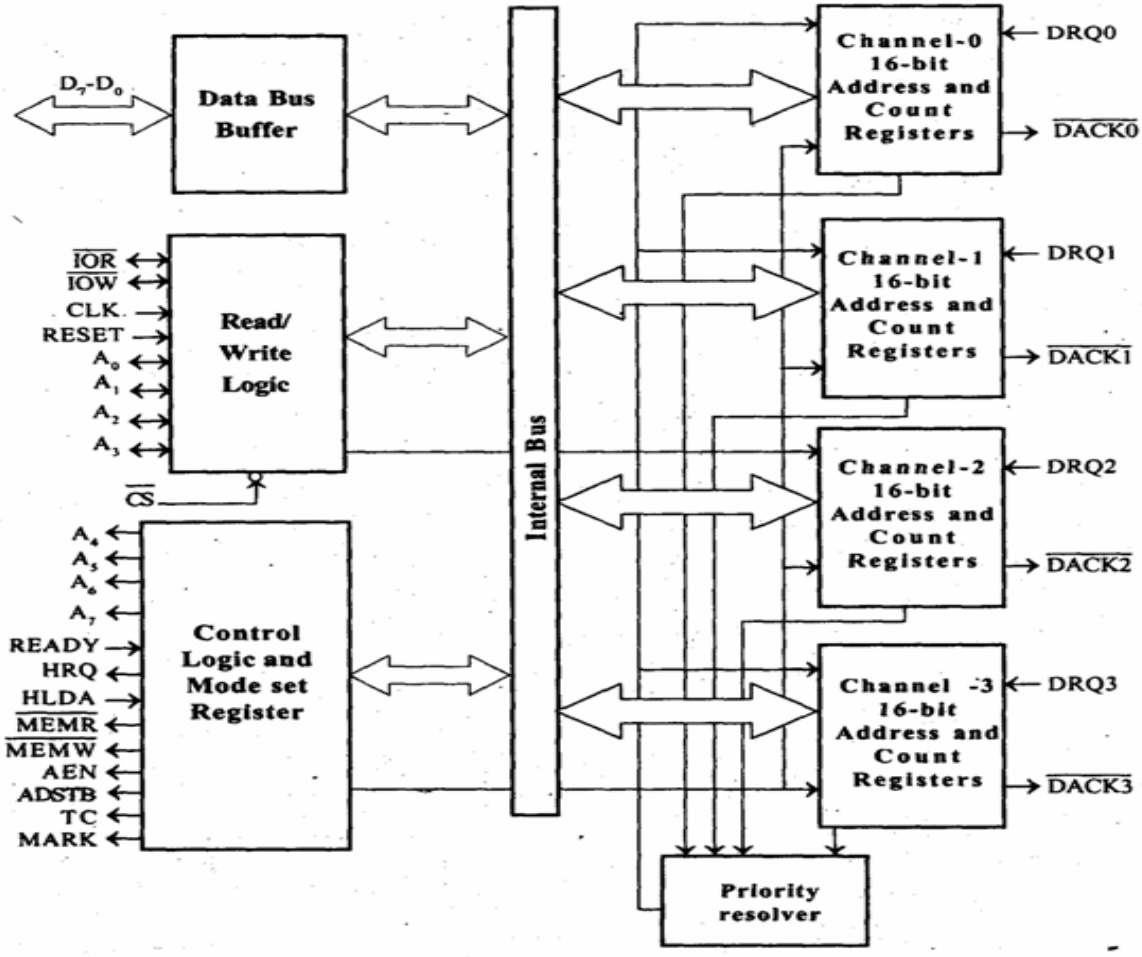
How DMA Operations are Performed?

Following is the sequence of operations performed by a DMA .

- Initially, when any device has to send data between the device and the memory, the device has to send DMA request (DRQ) to DMA controller.
- The DMA controller sends Hold request (HRQ) to the CPU and waits for the CPU to assert theHLDA.
- Then the microprocessor tri-states all the data bus, address bus, and control bus.

The CPU leaves the control over bus and acknowledges the HOLD request through HLDA signal.

- Now the CPU is in HOLD state and the DMA controller has to manage the operations over buses between the CPU, memory, and I/O devices.
- The chip support four DMA channels, i.e. four peripheral devices can independently request for DMA data transfer through these channels at a time.
-



Internal Architecture of 8257

The DMA controller has

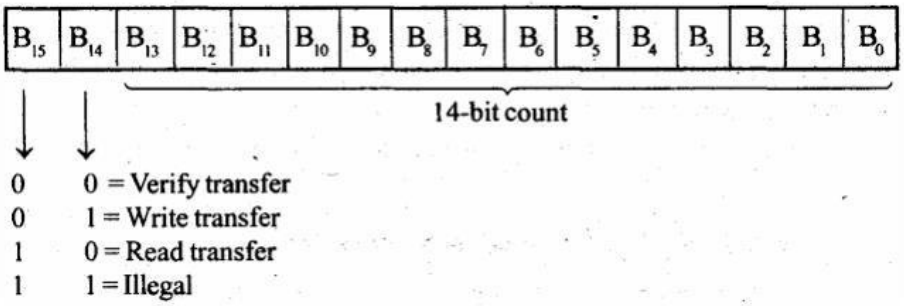
- 8-bit internal databuffer,
- a read/writeunit,
- a controlunit,
- a priority resolving unit along with a set ofregisters.

The 8257 performs the DMA operation over **four independent DMA channels.**

- Each of four channels of 8257 has a pair of **two 16-bit registers,** viz. DMA **address register and terminal countregister.**
- There are two common registers for all the channels, namely, **mode set register and status register**. Thus there are a total of tenregisters.

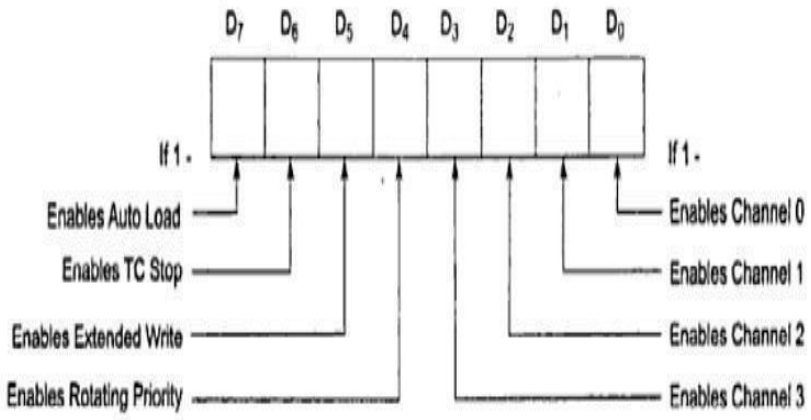
*The CPU selects one of these **ten registers using address lines Ao-A3.** Table shows how the Ao-A3 bits may be used for selecting one of these registers.*

Register	Address			
	A_3	A_2	A_1	A_0
Channel-0 DMA address register	0	0	0	0
Channel-0 Count register	0	0	0	1
Channel-1 DMA address register	0	0	1	0
Channel-1 Count register	0	0	1	1
Channel-2 DMA address register	0	1	0	0
Channel-2 Count register	0	1	0	1
Channel-3 DMA address register	0	1	1	0
Channel-3 Count register	0	1	1	1
Mode set register (Write only)	1	0	0	0
Status register (Read only)	1	0	0	0



Mode Set Register

- The mode set register is used for programming the 8257 as per the requirements of the system.
- The function of the mode set register is to enable the **DMA channels individually and also to set the various modes of operation.**



- **Burst/Block/ continuous transfer** –
- When the DMAC operates in burst mode, the CPU is halted for the duration of the data transfer.
- If more than one channel requests service simultaneously, this transfer occur. The continuous transfer may be interrupted by an external device by pulling down the HLDA signal
- **Cyclic Stealing** -

  An alternative method in which DMA controller transfers one word at a time after which it must return the control of the buses to the CPU. The CPU delays its operation only for one memory cycle to allow the direct memory I/O transfer to "steal" one memory cycle.

Priority Resolver

Fixed Priority - In 8257, if each device connected to a channel is assigned fixed priority scheme

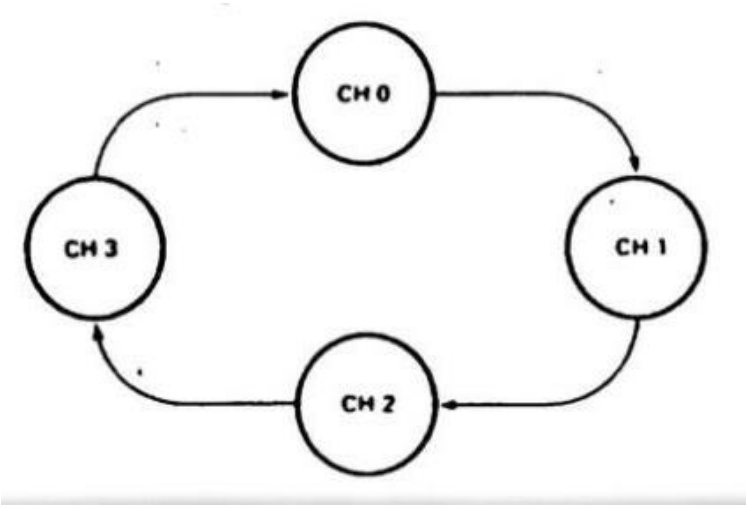
DRQ0 has the highest priority.

DRQ1

DRQ2

DRQ3 has the least priority.

Rotating Priority - The priority of the channels varies frequently. The channel which has been serviced has lowest priority



PROGRAMMING AND APPLICATIONS CASE STUDIES

TRAFFIC LIGHT CONTROL

Statement:

Design a microprocessor system to control traffic lights. The traffic light arrangement is as shown in Fig. The traffic should be controlled in the following manner.

Allow traffic from W to E and E to W transition for 20 seconds. 2) Give transition period of 5 seconds (Yellow bulbs ON) 3) Allow traffic from N to 5 and 5 to N for 20 seconds 4) Give transition period of 5 seconds (Yellow bulbs ON) 5) Repeat the process.

HARDWARE FOR TRAFFIC LIGHT CONTROL

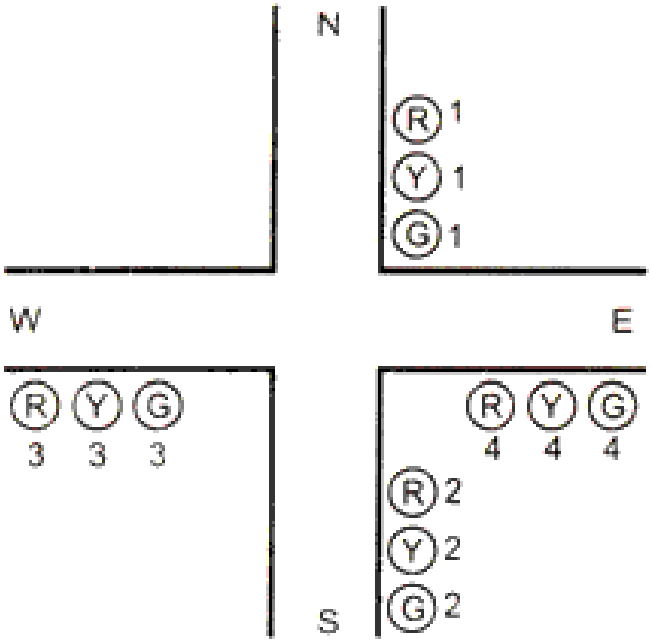
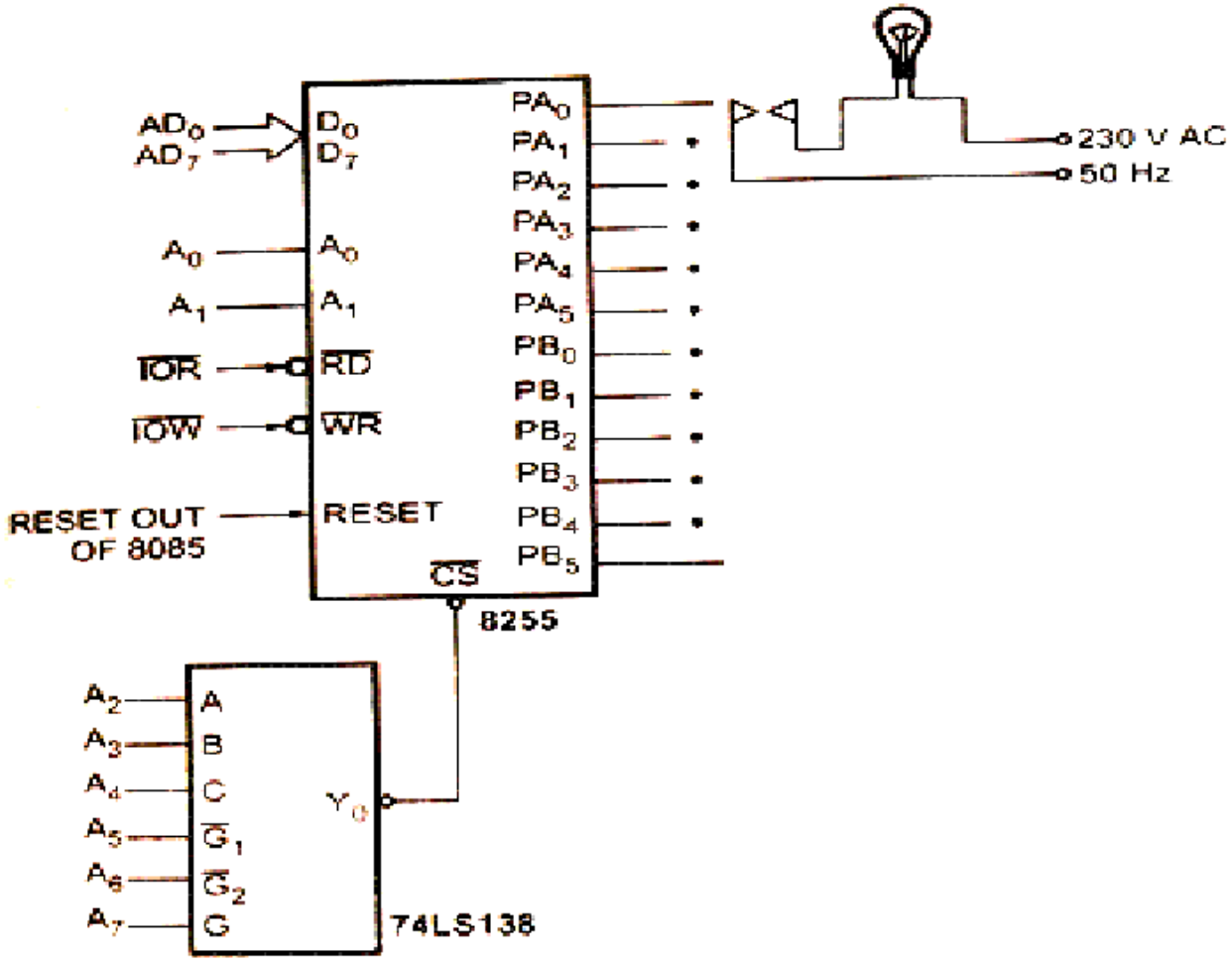


Fig. shows the interfacing diagram to control 12 electric bulbs. Port A is used to control lights on N-S road and Port B is used to control lights on W-E road. Actual pin connections are listed in Table 1 below.

Pins	Light	Pins	Light
PA_0	R_1	PB_0	R_3
PA_1	Y_1	PB_1	Y_3
PA_2	G_1	PB_2	G_3
PA_3	R_2	PB_3	R_4
PA_4	Y_2	PB_4	Y_4
PA_5	G_2	PB_5	G_4

The electric bulbs are controlled by relays. The 8255 pins are used to control relay on-off action with the help of relay driver circuits. The driver circuit includes 12 transistors to drive 12 relays. Fig. also shows the interfacing of 8255 to the system.

INTERFACING DIAGRAM



Ports / Control Register	Address lines								Address
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Port A	1	0	0	0	0	0	0	0	80H
Port B	1	0	0	0	0	0	0	1	81H
Port C	1	0	0	0	0	0	1	0	82H
Control Register	1	0	0	0	0	0	1	1	83H

SOFTWARE FOR TRAFFIC LIGHT CONTROL

Control word : For initialization of 8255.

BSR/IO	$MODE_A$		P_A	PC_H	MODE B	P_B	PC_L
1	0	0	0	X	0	0	X

= 80H

Fig. Control word

Table shows the data bytes to be sent for specific combinations.

To glow	PB_7	PB_6	PB_5	PB_4	PB_3	PB_2	PB_1	PB_0	PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0	Port B Output	Port A Output
R_1,R_2,G_3 and G_4	X	X	1	0	0	1	0	0	X	X	0	0	1	0	0	1	24H	09H
Y_1,Y_2,Y_3 and Y_4	X	X	0	1	0	0	1	0	X	X	0	1	0	0	1	0	12H	12H
R_3,R_4,G_1 and G_2	X	X	0	0	1	0	0	1	X	X	1	0	0	1	0	0	09H	24H

Source program:

```
    MVI A, 80H          : Initialize 8255, port A and port B
    OUT 83H (CR)        : in output mode
START: MVI A, 09H
    OUT 80H (PA)        : Send data on PA to glow R1 and R2
    MVI A, 24H
```

```
    OUT 81H (PB)        : Send data on PB to glow G3 and G4
    MVI C, 28H          : Load multiplier count (40₁₀) for delay
    CALL DELAY          : Call delay subroutine
    MVI A, 12H
    OUT (80H) PA        : Send data on Port A to glow Y1 and Y2
    OUT (81H) PB        : Send data on port B to glow Y3 and Y4
    MVI C, 0AH          : Load multiplier count (10₁₀) for delay
CALL: DELAY             : Call delay subroutine
    MVI A, 24H
    OUT (80H) PA        : Send data on port A to glow G1 and G2
    MVI A, 09H
    OUT (81H) PB        : Send data on port B to glow R3 and R4
    MVI C, 28H          : Load multiplier count (40₁₀) for delay
    CALL DELAY          : Call delay subroutine
    MVI A, 12H
    OUT PA              : Send data on port A to glow Y1 and Y2
    OUT PB              : Send data on port B to glow Y3 and Y4
    MVI C, 0AH          : Load multiplier count (10₁₀) for delay
    CALL DELAY          : Call delay subroutine
    JMP START
```

Delay Subroutine:

```
DELAY: LXI D, Count     : Load count to give 0.5 sec delay
BACK: DCX D             : Decrement counter
    MOV A, D
    ORA E               : Check whether count is 0
    JNZ BACK            : If not zero, repeat
    DCR C               : Check if multiplier zero, otherwise repeat
    JNZ DELAY
    RET                 : Return to main program
```

SEVEN SEGMENT DISPLAY INTERFACING USING 8085 µP

Statement:

Interface an 8-digit 7 segment LED display using 8255 to the 8085 microprocessor system and write an 8085 assembly language routine to display message on the display.

HARDWARE FOR EIGHT DIGIT SEVEN SEGMENT DISPLAY INTERFACE

Fig. shows the multiplexed eight 7-segment display connected in the 8085 system using 8255. In this circuit port A and port B are used as simple latched output ports. Port A provides the segment data inputs to the display and port B provides a means of selecting a display position at a time for multiplexing the displays. A0-A7 lines are used to decode the addresses for 8255. For this circuit different addresses are:

PA = 00H PB = 01H

PC = 02H CR = 03H.

The register values are chosen in Fig. such that the segment current is 80 mA. This current is required to produce an average of 10 mA per segment as the displays are multiplexed. In this type of display system, only one of the eight display position is 'ON' at any given instant. Only one digit is selected at a time by giving low signal on the corresponding control line. Maximum anode current is 560 mA (7-segments x 80 mA = 560 mA), but the average anode current is 70 mA.

HARDWARE FOR EIGHT DIGIT SEVEN SEGMENT DISPLAY INTERFACE

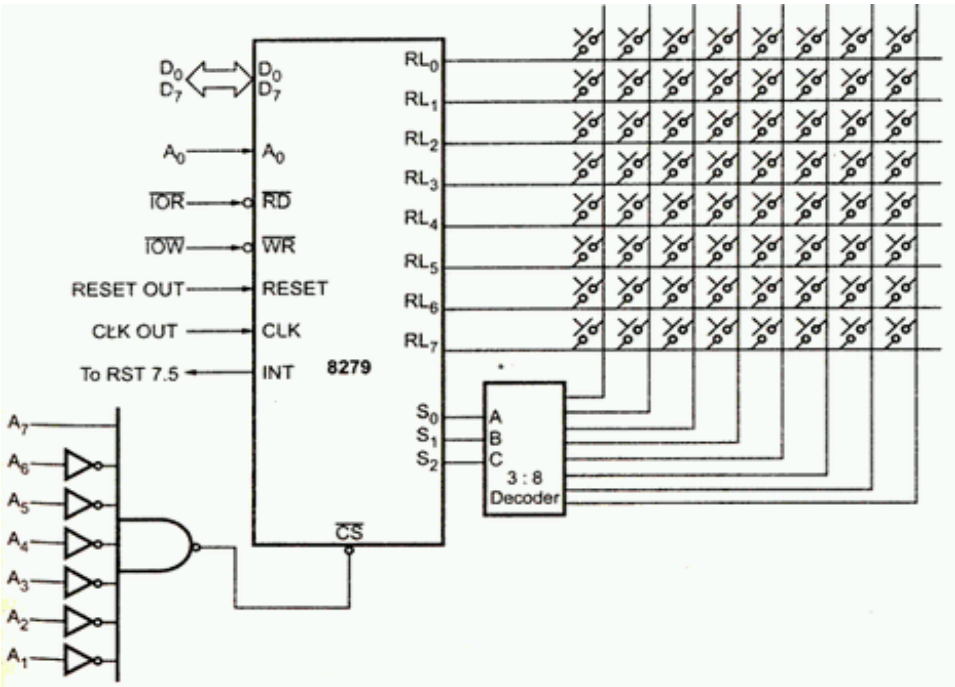


Fig. — Interfacing of 8 × 8 matrix keyboard in interrupt driven keyboard mode

SOFTWARE FOR EIGHT DIGIT SEVEN SEGMENT DISPLAY INTERFACE

For 8255, Port A and B are used as output ports. The control word format of 8255 according to hardware connections is:

BSR	Mode A		P_A	P_{CU}	Mode B	P_B	P_{CL}	
1	0	0	0	X	0	0	X	= 80 H

Fig. — Control word format for 8255

Source program:

SOFTWARE TO INITIALIZE 8255:

```
MVI     A, 80H          : Load control word in AL
OUT     CR              : Load control word in CR
```

SUBROUTINE TO DISPLAY MESSAGE ON MULTIPLEXED LED DISPLAY:

SET UP REGISTERS FOR DISPLAY:

```
    MVI B, 08H          : load count
    MVI C, 7FH          : load select pattern
    LXI H, 6000B        : starting address of message
```

DISPLAY MESSAGE:

```
DISP 1: MOV A, C        : select digit
    OUT PB
    MOV A, M            : get data
    OUT PA              : display data
    CALL DELAY          : wait for some time
DISP 1: MOV A, C
    RRC
    MOV C, A            : adjust selection pattern
    INX H
    DCR B               : Decrement count
    JNZ DISP 1          : repeat 8 times
    RET
```

Delay subroutine:

```
Delay: LXI D, Count
Back: DCX D
    MOV A, D
    ORA E
    JNZ Back
    RET
```

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF SCIENCE AND HUMANITIES

DEPARTMENT OF PHYSICS

UNIT - V

UNIT – I–Microprocessor and Microcontroller – SPH1313

MICROCONTROLLER

A microcontroller is a highly integrated single chip, which consists of on chip CPU (Central Processing Unit), RAM (Random Access Memory), EPROM/PROM/ROM (Erasable Programmable Read Only Memory), I/O (input/output) – serial and parallel, timers, interrupt controller. For example, Intel 8051 is 8-bit microcontroller and Intel 8096 is 16-bit microcontroller.

TYPES OF MICROCONTROLLERS

Microcontrollers can be classified on the basis of internal bus width, architecture, memory and instruction set as 4-bit, 8-bit, 16-bit and 32-bit microcontrollers.

Difference between microprocessor and microcontroller?

1. Key difference in both of them is presence of external peripheral, where microcontrollers have RAM, ROM, EEPROM embedded in it while we have to use external circuits in case of microprocessors.

2. Asalltheperipheralofmicrocontrollerareonsinglechipitiscompactwhilemicroprocessoris bulky.

3. Microcontrollers are made by using complementary metal oxide semiconductor technology so theyarefarcheaperthanmicroprocessors.Inadditiontheapplicationsmadewithmicrocontrolle rs arecheaperbecausetheyneed lesserexternalcomponents,whiletheoverallcostofsystemsmade with microprocessors are high because of the high number of external components required for suchsystems.

4. Processing speed of microcontrollers is about 8 MHz to 50 MHz, but in contrary processing speed of general microprocessors is above 1 GHz so it works much faster thanmicrocontrollers.

5. Generally, microcontrollers have power saving system, like idle mode or power saving mode so overall it uses less power and also since external components are low overall consumption of power is less. While in microprocessors generally there is no power saving system and also many external components are used with it, so its power

consumption is high in comparison with microcontrollers.

6. Microcontrollersarecompactsoitmakesthemfavorableandefficientsystemforsmallproducts and applications while microprocessors are bulky so they are preferred for largerapplications.

7. Tasks performed by microcontrollers are limited and generally less complex. While task performedbymicroprocessorsaresoftwaredevelopment,Gamedevelopment,website,documents making etc. which are generally more complex so require more memory and speed so that's why external ROM, RAM are used withit.

8. Microcontrollers are based on Harvard architecture where program memory and data memory areseparate whilemicroprocessorsarebasedonvonNeumannmodelwhereprogramanddataare stored in same memorymodule.

8051 Microcontrollers Architecture

- 8051 microcontrollersis designed by Intel in1981.
- It is an 8-bitmicrocontroller.
- It is built with 40 pins DIP (dual inlinepackage)
- 4kb of ROM storage and 128 bytes of RAMstorage
- 2 16-bittimers.
- Itconsistsofarefour parallel8-bitports,whichareprogrammableaswellasaddressable as per therequirement.
- An on-chip crystal oscillator is integrated in the microcontroller having crystal frequency of 12MHz.

In the diagram, the system bus connects all the support devices to the CPU. The system bus consists of an 8-bit data bus, a 16-bit address bus and bus control signals. All other devices like program memory, ports, data memory, serial interface, interrupt control, timers, and the CPU are all interfaced together through the system bus.

1. Central Processing Unit(CPU):

The CPU of 8051 Architecture consists of eight-bit Arithmetic and Logic unit with associated registers like A, B, PSW, SP, the sixteen bit program counter and "Data pointer" (DPTR) registers.

2. ALU:

It is 8 bit unit. It performs arithmetic operation as addition, subtraction, multiplication, division, increment and decrement. It performs logical operations like AND, OR and EX-OR. It performs compare, rotate and compliment operations. 8051 micro controller contains 34 general purpose registers or working registers. Two of them are called math registers A & B and 32 are bank of registers.

a. Accumulator(A-reg):

It is 8 bit register. Its address is E0H and it is bit and byte accessible. Result of arithmetic & logic operations performed by ALU is accumulated by this register. Therefore it is called accumulator register. It is used to store 8 bit data and to hold one of operand of ALU units during arithmetical and logical operations. Most of the instructions are carried out on accumulator data.

b. B-register:

It is special 8 bit math register. It is bit and byte accessible. It is used in conjunction with A register as I/P operand for ALU. It is used as general purpose register to store 8 bit data.

c.

:

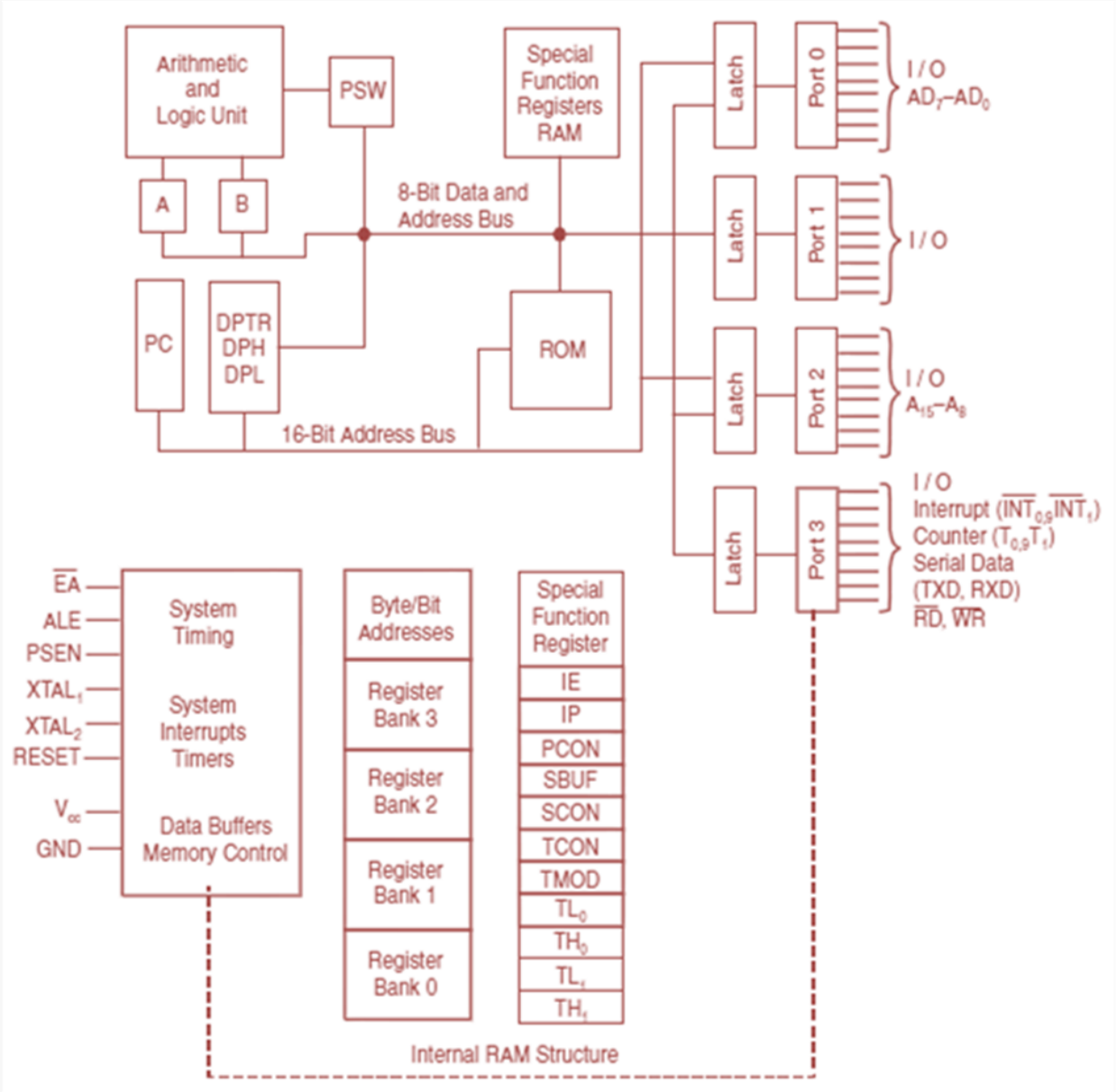


Figure 1. Internal of Architecture of 8051

d. PSW:

It is 8 bit register. Its address is D0H and It is bit and byte accessible. It has 4 conditional flags or math flags which sets or resets according to condition of result. It has 3 control flags, by setting or resetting bit required operation or function can be achieved. The format of flag register is as shown below

CY	CA	F0	RS1	RS0	OV	-	P

CY	Carry Flag
AC	Auxiliary Carry Flag
F0	Flag 0 available to user for general purpose.
RS1	Register Bank selector bit 1
RS0	Register Bank selector bit 0
OV	Overflow Flag
-	User definable FLAG
P	Parity FLAG. Set/ cleared by hardware during instruction cycle to indicate even/odd number of 1 bit in accumulator.

i. MATHFLAG:

1. Carry Flag(CY): During addition and subtraction any carry or borrow is generated then carry flag is set otherwise carry flag resets. It is used in arithmetic, logical, jump, rotate and Booleanoperations.

2. Auxiliary carry flag(AC): If during addition and subtraction any carry or borrowis generated from lower 4 bit to higher 4 bit then AC sets else it resets. It is used in BCD arithmetic operations.

3. Overflow flag(OV): If in signed arithmetic operations result exceeds more than 7 bit than OV flag sets else resets.It is used in signed arithmetic operationsonly.

4. Parity flag(P): If in result, even no. Of ones "1" are present than it is called even parity and parity flag sets. In result odd no. Of ones "1"are present than it is called odd parity and parity flagresets.

ii. CONTROLFLAGS:

1. FO: It is user defined flag. The user defines the function of this flag. The user canset

,test n clear this flag through software.

RS1 and RS0: These flags are used to select bank of register by resetting thoseflags which are as shown in table:

RS1	RS2	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

2.Program counter(PC): The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute is found in memory. It is used to hold 16 bit address of internal RAM, external RAM or external ROM locations. When the 8051 is initialized PC always starts at 0000h and is incremented each time an instruction is executed. It is important to note that PC isnt always incremented by one and neverdecremented.

3. Data pointer register(DTPR): It is a 16 bit register used to hold address of external or internal RAM where data is stored or result is to be stored. It is used to store 16 bit data. It is divided into2- 8bit registers, DPH-data pointer higher order (83H) and DPL-data pointer lower order (82H). Each register can be used as general purpose register to store 8 bit data and can also be used as memorylocation.

4. Stack pointer(SP): It is 8-bit register. It is byte addressable. Its address is 81H. It is used to hold the internal RAM memory location addresses which are used as stack memory. When the data is to be placed on stack by push instruction, the content of stack pointer is incremented by 1, and when data is retrieved from stack, content of stack of stack pointer is decremented by1.

iii. Special function Registers(SFR): The 8051 microcontroller has 11 SFR divided in 4groups:

A.

B. Timer/Counter register: 8051 microcontroller has 2-16 bit Timer/counter registers called Timer-reg-T0 And Timer/counter Reg-T1.Each register is 16 bit register divide into lower and higher byte register as shown below: These register are used to hold initial no. of count. All of the 4 register are byteaddressable.

1. Timer control register: 8051 microcontroller has two 8-bit timer control register i.e. TMOD and TCON register. TMOD Register: it is 8-bit register. Its address is 89H. It is byte addressable. It used to select mode and control operation of time by writing control word.

2. TCON register: It is 8-bit register. Its address is 88H. It is byte addressable. Its MSB 4-bit are used to control operation of timer/ counter and LSB 4-bit are used for external interruptcontrol.

C. Serial data register: 8051 micro controller has 2 serial data register viz. SBUF andSCON.

1. Serial buffer register (SBUF): it is 8-bit register. It is byte addressable .Its address is 99H. It is used to hold data which is to be transferredserially.

2. Serial control register (SCON): it is 8-bit register. It is bit/byte addressable. Its address is 98H. The 8-bit loaded into this register controls the operation of serial communication.

D. Interrupt register: 8051 µC has 2 8-bit interruptregister.

1. Interrupt enable register (IE): it is 8-bit register. It is bit/byte addressable. Its address is A8H.it is used to enable and disable function ofinterrupt.

2. Interrupt priority register (IP): It is 8-bit register. It is bit/byte addressable. Its address is B8H. it is used to select low or high level priority of each individualinterrupts.

E. Power control register (PCON): it is 8-bit register. It is byte addressable .Its address is 87H. its bits are used to control mode of power saving circuit, either idle or power

down mode and also one bit is used to modify baud rate of serialcommunication.

Internal RAM

Internal RAM has memory 128-byte. See 8051 hardware for further internal RAM design. Internal RAM is organized into three distinct areas: 32 bytes working registers from address 00h to 1Fh 16 bytes bit addressable occupies RAM byte address 20h to 2Fh, altogether 128 addressable bits General purpose RAM from 30h to 7Fh.

Internal ROM

Data memory and program code memory both are in different physical memory but both have the same addresses. An internal ROM occupied addresses from 0000h to 0FFFh. PC addresses program codes from 0000h to 0FFFh. Program addresses higher than 0FFFh that exceed the internal ROM capacity will cause 8051 architecture to fetch codes bytes from external program memory.

The pin diagram of 8051 microcontroller

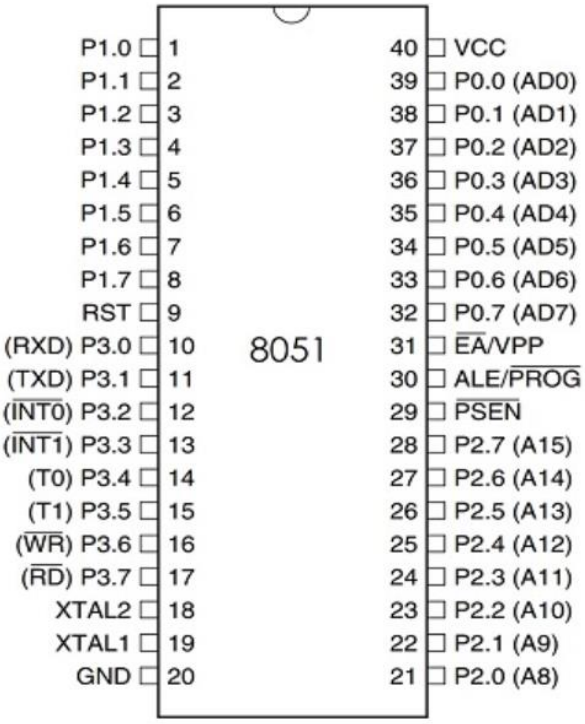


Figure 2. Pin configuration of 8051

Pins 1 to 8 − These pins are known as Port 1. This port doesn't serve any other functions. It is internally pulled up, bi-directional I/O port.

Pin 9 − It is a RESET pin, which is used to reset the microcontroller to its initial values.

Pins 10 to 17 − These pins are known as Port 3. This port serves some functions like interrupts, timer input, control signals, serial communication signals RxD and TxD, etc.

Pins 18 & 19 − These pins are used for interfacing an external crystal to get the system clock.

Pin 20 − This pin provides the power supply to the circuit.

Pins 21 to 28 − These pins are known as Port 2. It serves as I/O port. Higher order address bus signals are also multiplexed using this port.

Pin 29 − This is PSEN pin which stands for Program Store Enable. It is used to read a signal from the external program memory.

Pin 30 − This is EA pin which stands for External Access input. It is used to enable/disable the external memory interfacing.

Pin 31 − This is ALE pin which stands for Address Latch Enable. It is used to demultiplex the address-data signal of port.

Pins 32 to 39 − These pins are known as Port 0. It serves as I/O port. Lower order address and data bus signals are multiplexed using this port.

Pin 40 − This pin is used to provide power supply to the circuit.

8051 Microcontroller Special Function Registers (SFRs)

The internal RAM or Data Memory of the 8051 Microcontroller is divided in to General Purpose Registers, Bit Addressable Registers, Register Banks and Special Function Registers or SFRs.

The 8051 Microcontroller Special Function Registers are used to program and control different hardware peripherals like Timers, Serial Port, I/O Ports etc. In fact, by manipulating the 8051 Microcontroller Special Function Registers (SFRs), you can assess or change the operating mode of the 8051 Microcontroller.

Basic structure of 8051 Microcontroller's Internal RAM.

Special Function Registers act as a control table that monitor and control the operation of the 8051 Microcontroller. If you observe in Internal RAM Structure, the Address Space from 80H to FFH is allocated to SFRs.

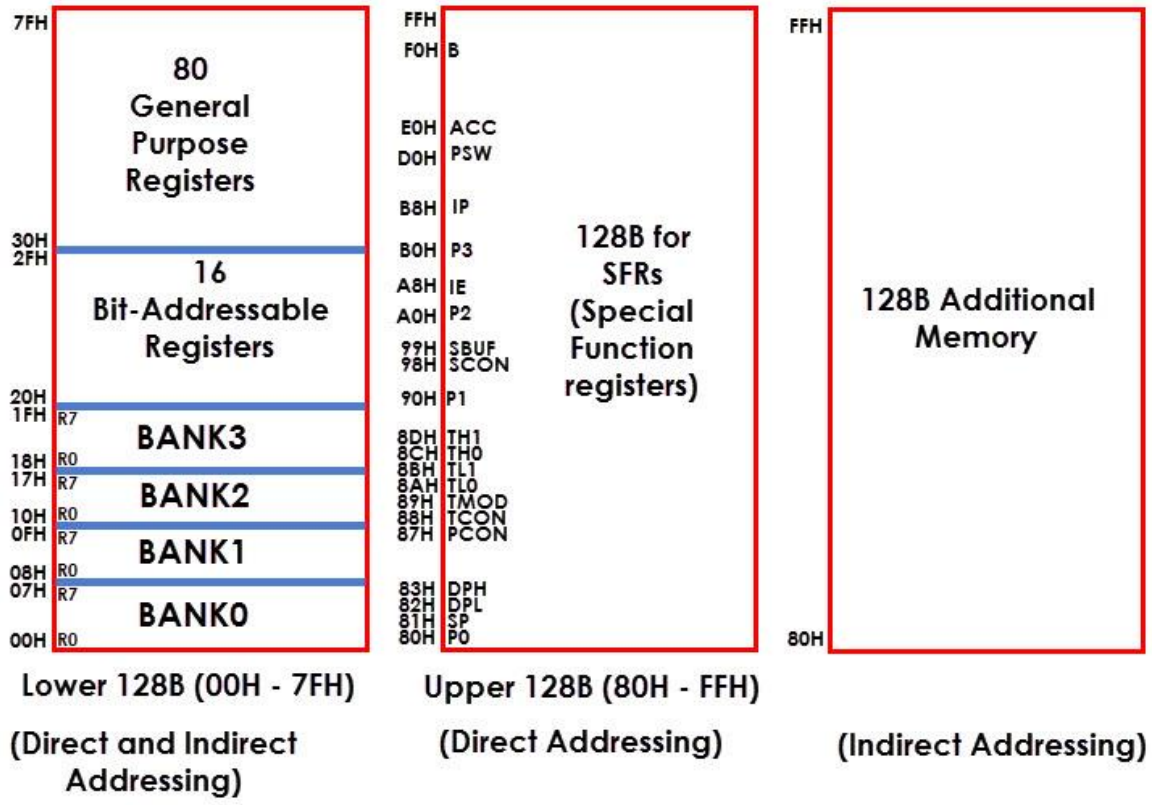


Figure 3. Memory Organization of 8051

Since the SFRs are a part of the Internal RAM Structure, you can access SFRs as if you access the Internal RAM. The main difference is the address space: first 128 Bytes (00H to 7FH) is for regular Internal RAM and next 128 Bytes (80H to FFH) is for SFRs.

Out of these 128 Memory Locations (80H to FFH), there are only 21 locations that are actually assigned to SFRs. Each SFR has one Byte Address and also a unique name which specifies its purpose.

The 21 Special Function Registers of 8051 Microcontroller are categorized in to seven groups.

They are:

Math or CPU Registers: A and B

Table. 1 8051 Microcontroller Special Function Registers

Name of the Register	*Function*	*Internal RAM Address (HEX)*
ACC	Accumulator	E0H
B	B Register (for Arithmetic)	F0H
DPH	Addressing External Memory	83H
DPL	Addressing External Memory	82H
IE	Interrupt Enable Control	A8H
IP	Interrupt Priority	B8H
P0	PORT 0 Latch	80H
P1	PORT 1 Latch	90H
P2	PORT 2 Latch	A0H
P3	PORT 3 Latch	B0H
PCON	Power Control	87H
PSW	Program Status Word	D0H
SCON	Serial Port Control	98H
SBUF	Serial Port Data Buffer	99H
SP	Stack Pointer	81H
TMOD	Timer / Counter Mode Control	89H
TCON	Timer / Counter Control	88H
TL0	Timer 0 LOW Byte	8AH
TH0	Timer 0 HIGH Byte	8CH
TL1	Timer 1 LOW Byte	8BH
TH1	Timer 1 HIGH Byte	8DH

Status Register: PSW (Program Status Word)

Pointer Registers: DPTR (Data Pointer – DPL, DPH) and SP (Stack Pointer)

I/O Port Latches: P0 (Port 0), P1 (Port 1), P2 (Port 2) and P3 (Port 3)

Peripheral Control Registers: PCON, SCON, TCON, TMOD, IE and IP

Peripheral Data Registers: TL0, TH0, TL1, TH1 and SBUF

CPU or Math Registers

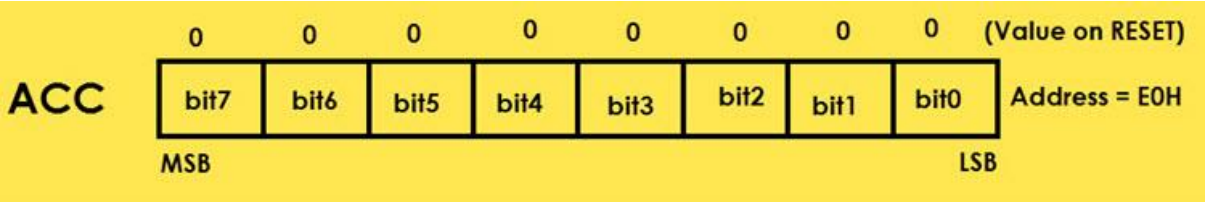
A or Accumulator (ACC)

The Accumulator or Register A is the most important and most used 8051 Microcontroller SFRs. The Register A is located at the address E0H in the SFR memory space. The Accumulator is used to hold the data for almost all the ALU Operations.

Some of the operations where the Accumulator is used are:

- Arithmetic Operations like Addition, Subtraction, Multiplication etc.
- Logical Operations like AND, OR, NOT etc.
- Data Transfer Operations (between 8051 and External Memory)

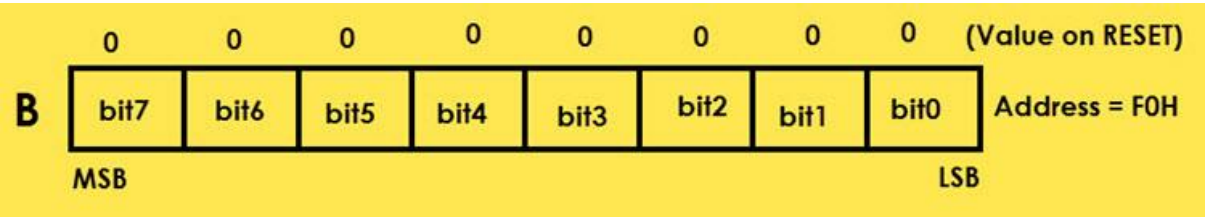
The results of all Arithmetic and Logical operations are stored in Accumulator.



B (Register B)

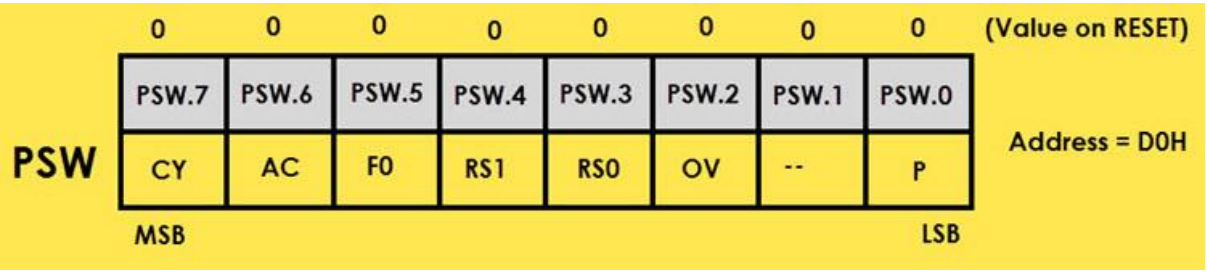
The B Register is used along with the ACC in Multiplication and Division operations. These two operations are performed on data that are stored only in Registers A and B. During Multiplication Operation, one of the operand (multiplier or multiplicand) is stores in B Register and also the higher byte of the result.

Register B is located at the address F0H of the SFR Address Space.



Program Status Word (PSW)

The PSW or Program Status Word Register is also called as Flag Register and is one of the important SFRs. The PSW Register consists of Flag Bits, which help the programmer in checking the condition of the result and also make decisions.



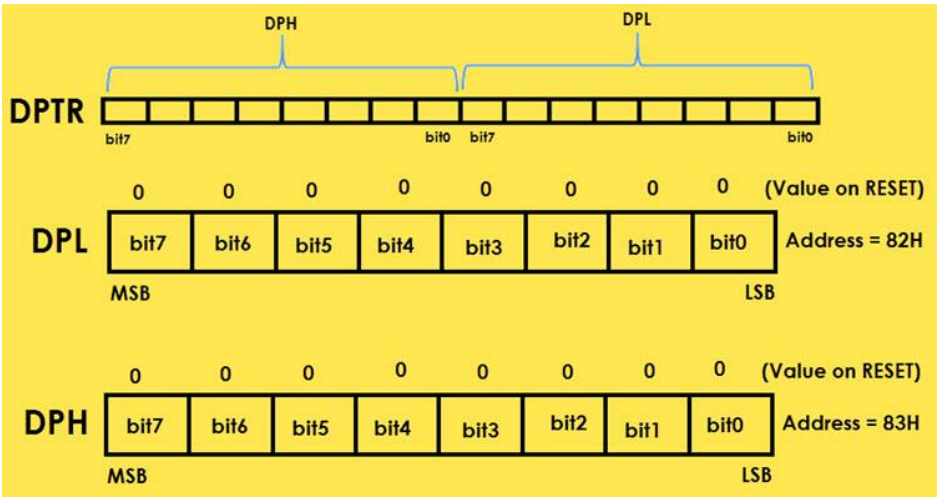
Function of each flag.

BIT	SYMBOL	FLAG NAME			DESCRIPTION
7	C or CY	Carry			Used in Arithmetic, Logic & Boolean Operations
6	AC	Auxiliary Carry			Used in BCD Arithmetic
5	F0	Flag 0			General Purpose User Flag
4	RS1	Register Bank Selection Bit 1			
3	RS0	Register Bank Selection Bit 1			
		RS1	RS0	Bank	
		0	0	Bank 0	
		0	1	Bank 1	
		1	0	Bank 2	
		1	1	Bank 3	
2	OV	Overflow			Used in Arithmetic Operations
1	--	Reserved			May be used as a General Purpose Flag
0	P	Parity			Set to 1 if A has odd # of 1's; otherwise Reset

Pointer Registers

Data Pointer (DPTR – DPL and DPH)

The Data Pointer is a 16-bit Register. The Data Pointer can be used as a single 16-bit register (as DPTR) or two 8-bit registers (as DPL and DPH).



DPTR doesn't have a physical Memory Address but the DPL (Lower Byte of DPTR) and DPH (Higher Byte of DPTR) have separate addresses in the SFR Memory Space. DPL = 82H and DPH = 83H.

The DPTR Register is used by the programmer addressing external memory (Program – ROM or Data – RAM).

Stack Pointer (SP)

SP or Stack Pointer points out to the top of the Stack and it indicates the next data to be accessed. Stack Pointer can be accesses using PUSH, POP, CALL and RET Instructions. The Stack Pointer is an 8-bit register and upon reset, the Stack Pointer is initialized with 07H.

When writing a new data byte into the stack, the SP (Stack Pointer) is automatically incremented by 1 and the new data is written at an address SP+1. When reading data from stack, the data is retrieved from the Address in SP and after that the SP is decremented by 1 (SP-1).

I/O Port Registers (P0, P1, P2 and P3)

The 8051 Microcontroller four Ports which can be used as Input and/or Output. These four ports are P0, P1, P2 and P3. Each Port has a corresponding register with same names (the Port Registers are lso P0, P1, P2 and P3). The addresses of the Port Registers are as follows: P0 – 80H, P1 – 90H, P2 – A0H and P2 – B0H.

Peripheral Control Registers

PCON (Power Control)

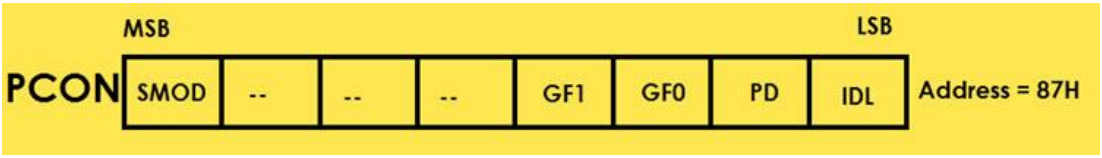
The PCON or Power Control register, as the name suggests is used to control the 8051 Microcontroller's Power Modes and is located at 87H of the SFR Memory Space. Using two bits in the PCON Register, the microcontroller can be set to Idle Mode (IDL) and Power Down Mode (PD).

During Idle Mode, the Microcontroller will stop the Clock Signal to the ALU (CPU) but it is given to other peripherals like Timer, Serial, Interrupts, etc. In order to terminate the Idle Mode, you have to use an Interrupt or Hardware Reset.

In the Power Down Mode, the oscillator will be stopped and the power will be reduced to 2V. To terminate the Power Down Mode (PD), you have to use the Hardware Reset.

Apart from these two, the PCON Register can also be used for few additional purposes. The SMOD Bit in the PCON Register is used to control the Baud Rate of the Serial Port.

There are two general purpose Flag Bits in the PCON Register, which can be used by the programmer during execution.

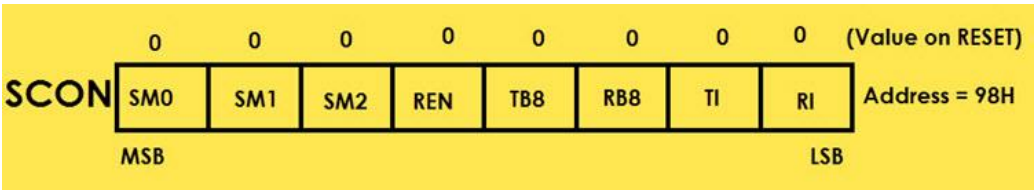


SMOD is used to decide the BAUD rate in serial operating modes 1,2 and 3.

SCON (Serial Control)

The Serial Control or SCON SFR is used to control the 8051 Microcontroller's Serial Port. It is located as an address of 98H. Using SCON, you can control the Operation Modes of the Serial Port, Baud Rate of the Serial Port and Send or Receive Data using Serial Port.

SCON Register also consists of bits that are automatically SET when a byte of data is transmitted or received.



Serial Port Mode Control Bits

SM0	SM1	Mode	Description	Baud Rate
0	0	0	8-Bit Synchronous Shift Register Mode	Fixed Baud Rate (Frequency of oscillator / 12)
0	1	1	8-bit StandardUART mode	Variable Baud Rate (Can be set by Timer 1)
1	0	2	9-bit Multiprocessor Comm. mode	Fixed Baud Rate (Frequency of oscillator / 32 or Frequency of oscillator / 64
1	1	3	9-bit Multiprocessor Comm. mode	Variable Baud Rate (Can be set by Timer 1)

TMOD (Timer Mode)

The TMOD or Timer Mode register or SFR is used to set the Operating Modes of the Timers T0 and T1. The lower four bits are used to configure Timer0 and the higher four bits are used to configure Timer1.

The Gatex bit is used to operate the Timerx with respect to the INTx pin or regardless of the INTx pin.

GATEx = 1 ==> Allows external clock (Hardware interrupt)

GATEx = 0 ==> Does not allows external clock (Software interrupt)

C/Tx = 1 ==> Operates in Counter Mode

C/Tx = 0 ==> Operates in Timer Mode

Timer 16 bit TH(8 bits) and TL(8 bits)

TxM0	TxM1	Mode	Description
0	0	0	13-bit Timer Mode (THx – 8-bit and TLx – 5-bit)
0	1	1	16-bit Timer Mode
1	0	2	8-bit Auto Reload Timer Mode TH-Functions as a counter (Auto reload value) TL – Holds the initial value of TH
1	1	3	Two 8-bit Timer-0 Mode or Split Timer Mode(Timer-1 is stopped) TLO controls timer-0; THO controls timer-1

NOTE: x = 0 for Timer 0 and x = 1 for Timer 1.

TCON (Timer Control)

Timer Control or TCON Register is used to start or stop the Timers of 8051 Microcontroller. It also contains bits to indicate if the Timers has overflowed. The TCON SFR also consists of Interrupt related bits.

Timer (Software)

TF1 – Timer – 1 overflow flag

TR1 - Timer -1 run control bit

TF0 – Timer – 0 overflow flag

TR0 - Timer -0 run control bit

Timer (Hardware)

IE0 – Interrupt-0 edge flag

IT0 – Interrupt-0 type control bit

IE1 – Interrupt-1 edge flag

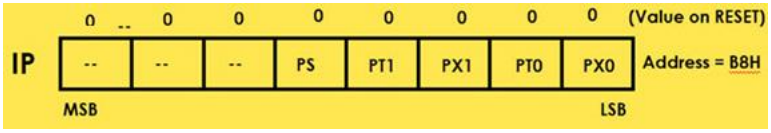
IT1 – Interrupt-1 type control bit

IE (Interrupt Enable)

The IE or Interrupt Enable Register is used to enable or disable individual interrupts. If a bit is SET, the corresponding interrupt is enabled and if the bit is cleared, the interrupt is disabled. The Bit7 of the IE register i.e. EA bit is used to enable or disable all the interrupts.

IP (Interrupt Priority)

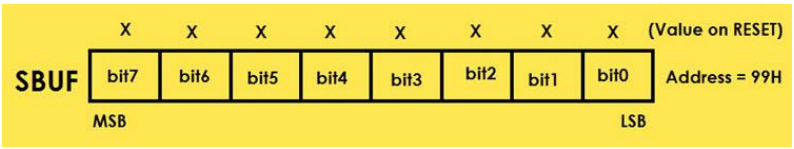
The IP or Interrupt Priority Register is used to set the priority of the interrupt as High or Low. If a bit is CLEARED, the corresponding interrupt is assigned low priority and if the bit is SET, the interrupt is assigned high priority.



Peripheral Data Registers

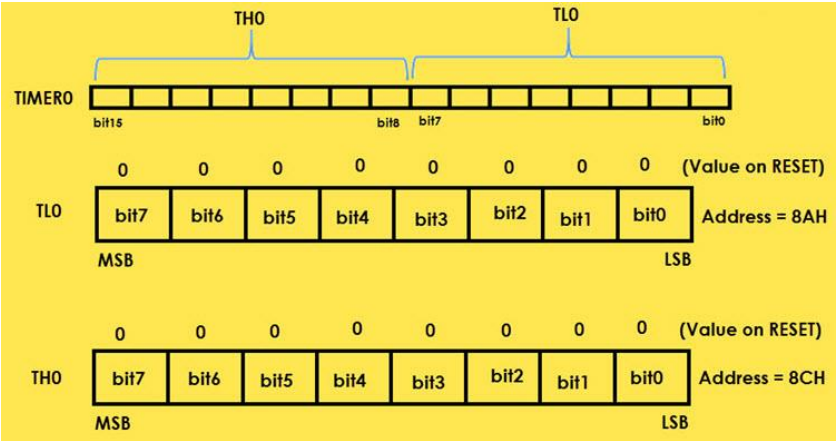
SBUF (Serial Data Buffer)

The Serial Buffer or SBUF register is used to hold the serial data while transmission or reception.



TL0/TH0 (Timer 0 Low/High)

The Timer 0 consists of two SFRs: TL0 and TH0. The TL0 is the lower byte and the TH0 is the higher byte and together they form a 16-bit Timer0 Register.



TL1/TH1 (Timer 1 Low/High)

The TL1 and TH1 are the lower and higher bytes of the Timer 0.

ADDRESSING MODES OF 8051

In 8051 there are 1-byte, 2-byte instructions and very few 3-byte instructions are present. The opcodes are 8-bit long. As the opcodes are 8-bit data, there are 256 possibilities. Among 256, 255 opcodes are implemented.

The clock frequency is12MHz, so 64 instruction types are executed in just 1 µs, and rest are just 2 µs. The Multiplication and Division operations take 4 µsto to execute.

In 8051 There are six types of addressing modes.

- Immediate AddressingMode
- Register AddressingMode
- Direct AddressingMode
- Register IndirectAddressing Mode
- Indexed AddressingMode
- Implied AddressingMode

Immediate addressing mode

In this Immediate Addressing Mode, the data is provided in the instruction itself. The data is provided immediately after the opcode. These are some examples of Immediate Addressing Mode.

```
MOVA,#0AFH;

MOVR3,#45H;

MOVDPTR,#FE00H;
```

In these instructions, the # symbol is used for immediate data. In the last instruction, there is DPTR. The DPTR stands for Data Pointer. Using this, it points the external data memory location. In the first instruction, the immediate data is AFH, but one 0 is added at the beginning. So when the data is starting with A to F, the data should be preceded by 0.

Register addressing mode

In the register addressing mode the source or destination data should be present in a register (R0 to R7). These are some examples of RegisterAddressing Mode.

```
MOVA, R5;

MOVR2,#45H;

MOVR0, A;
```

In 8051, there is no instruction like **MOVR5, R7**. But we can get the same result by using this instruction **MOV R5, 07H**, or by using **MOV 05H, R7**. But this two instruction will work when the selected register bank is **RB0**. To use another register bank and to get the same effect, we have to add the starting address of that register bank with the register number. For an example, if the RB2 is selected, and we want to access R5, then the address will be (10H + 05H = 15H), so the instruction will look like this **MOV 15H, R7**. Here 10H is the starting address of Register Bank 2.

Direct Addressing Mode

In the Direct Addressing Mode, the source or destination address is specified by using 8-bit data in the instruction. Only the internal data memory can be used in this mode. Here some of the examples of direct Addressing Mode.

```
MOV80H, R6;

MOVR2,45H;

MOVR0,05H;
```

The first instruction will send the content of registerR6 to port P0 (Address of Port 0 is 80H). The second one is forgetting content from 45H to R2. The third one is used to get data from Register R5 (When register bank RB0 is selected) to register R5.

Register indirect addressing Mode

In this mode, the source or destination address is given in the register. By using register indirect addressing mode, the internal or external addresses can be accessed. The R0 and R1 are used for 8-bit addresses, and DPTR is used for 16-bit addresses, no other registers can be used for addressing purposes. Let us see some examples of this mode.

```
MOV0E5H,@R0;

MOV@R1,80H
```

In the instructions, the @ symbol is used for register indirect addressing. In the first instruction, it is showing that theR0 register is used. If the content of R0 is 40H, then that instruction will take the data which is located at location 40H of the internal RAM. In the second one, if the content of R1 is 30H, then it indicates that the content of port P0 will be stored at location 30H in the internal RAM.

```
MOVXA,@R1;

MOV@DPTR, A;
```

In these two instructions, the X in MOVX indicates the external data memory. The external data memory can only be accessed in register indirect mode. In the first instruction if the R0 is holding 40H, then A will get the content of external RAM location40H. And in the second one, the content of A is overwritten in the location pointed by DPTR.

Indexed addressing mode

In the indexed addressing mode, the source memory can only be accessed from program memory only. The destination operand is always the register A. These are some examples of Indexed addressing mode.

```
MOVCA,@A+PC;

MOVCA,@A+DPTR;
```

The C in MOVC instruction refers to code byte. For the first instruction, let us consider A holds 30H. And the PC value is1125H. The contents of program memory location 1155H (30H + 1125H) are moved to register A.

Implied Addressing Mode

In the implied addressing mode, there will be a single operand. These types of instruction can work on specific registers only. These types of instructions are also known as register specific instruction. Here are some examples of Implied Addressing Mode.

```
RLA;

SWAPA;
```

These are 1- byte instruction. The first one is used to rotate the A register content to the Left. The second one is used to swap the nibbles in A.

TYPES OF INSTRUCTIONS IN 8051 MICROCONTROLLERS

INSTRUCTION SET

An 8051 Instruction consists of an Opcode (short of Operation – Code) followed by Operand(s) of size Zero Byte, One Byte or Two Bytes.

The Op-Code part of the instruction contains the Mnemonic, which specifies the type of operation to be performed. All Mnemonics or the Opcode part of the instruction are of One Byte size.

Coming to the Operand part of the instruction, it defines the data being processed by the instructions. The operand can be any of the following:

- No Operand
- Data value
- I/O Port
- Memory Location
- CPU register

There can multiple operands and the format of instruction is as follows:

MNEMONIC DESTINATION OPERAND, SOURCE OPERAND

A simple instruction consists of just the opcode. Other instructions may include one or more operands. Instruction can be one-byte instruction, which contains only opcode, or two-byte instructions, where the second byte is the operand or three byte instructions, where the operand makes up the second and third byte.Based on the operation they perform, all the instructions in the 8051 Microcontroller Instruction Set are divided into five groups. They are:

- Data Transfer Instructions
- Arithmetic Instructions
- Logical Instructions
- Boolean or Bit Manipulation Instructions

- Program Branching Instructions

Data Transfer Instructions

- The Data Transfer Instructions are associated with transfer with data between registers or external program memory or external data memory. The Mnemonics associated with Data Transfer are given below.

Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
MOV	A, #Data	A ← Data	Immediate	2	1
	A, Rn	A ← Rn	Register	1	1
	A, Direct	A ← (Direct)	Direct	2	1
	A, @Ri	A ← @Ri	Indirect	1	1
	Rn, #Data	Rn ← data	Immediate	2	1
	Rn, A	Rn ← A	Register	1	1
	Rn, Direct	Rn ← (Direct)	Direct	2	2
	Direct, A	(Direct) ← A	Direct	2	1
	Direct, Rn	(Direct) ← Rn	Direct	2	2
	Direct1, Direct2	(Direct1) ← (Direct2)	Direct	3	2
	Direct, @Ri	(Direct) ← @Ri	Indirect	2	2
	Direct, #Data	(Direct) ← #Data	Direct	3	2
	@Ri, A	@Ri ← A	Indirect	1	1
	@Ri, Direct	@Ri ← Direct	Indirect	2	2
	@Ri, #Data	@Ri ← #Data	Indirect	2	1
	DPTR, #Data16	DPTR ← #Data16	Immediate	3	2
MOVC	A, @A+DPTR	A ← Code Pointed by A+DPTR	Indexed	1	2
	A, @A+PC	A ← Code Pointed by A+PC	Indexed	1	2
	A, @Ri	A ← Code Pointed by Ri (8-bit Address)	Indirect	1	2
MOVX	A, @DPTR	A ← External Data Pointed by DPTR	Indirect	1	2
	@Ri, A	@Ri ← A (External Data 8-bit Addr)	Indirect	1	2
	@DPTR, A	@DPTR ← A (External Data 16-bit Addr)	Indirect	1	2
PUSH	Direct	Stack Pointer SP ← (Direct)	Direct	2	2
POP	Direct	(Direct) ← Stack Pointer SP	Direct	2	2
XCH	Rn	Exchange ACC with Rn	Register	1	1
	Direct	Exchange ACC with Direct Byte	Direct	2	1
	@Ri	Exchange ACC with Indirect RAM	Indirect	1	1
XCHD	A, @Ri	Exchange ACC with Lower Order Indirect RAM	Indirect	1	1

- *MOV*
- *MOVC*
- *MOVX*
- *PUSH*

- *POP*
- *XCH*
- *XCHD*

The following table lists out all the possible data transfer instruction along with other details like addressing mode, size occupied and number machine cycles it takes.

Arithmetic Instructions

Using Arithmetic Instructions, you can perform addition, subtraction, multiplication and division. The arithmetic instructions also include increment by one, decrement by one and a special instruction called Decimal Adjust Accumulator.

Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
ADD	A, #Data	A ← A + Data	Immediate	2	1
	A, Rn	A ← A + Rn	Register	1	1
	A, Direct	A ← A + (Direct)	Direct	2	1
	A, @Ri	A ← A + @Ri	Indirect	1	1
ADDC	A, #Data	A ← A + Data + C	Immediate	2	1
	A, Rn	A ← A + Rn + C	Register	1	1
	A, Direct	A ← A + (Direct) + C	Direct	2	1
	A, @Ri	A ← A + @Ri + C	Indirect	1	1
SUBB	A, #Data	A ← A – Data – C	Immediate	2	1
	A, Rn	A ← A – Rn – C	Register	1	1
	A, Direct	A ← A – (Direct) – C	Direct	2	1
	A, @Ri	A ← A – @Ri – C	Indirect	1	1
MUL	AB	Multiply A with B (A ← Lower Byte of A*B and B ← Higher Byte of A*B)	--	1	4
DIV	AB	Divide A by B (A ← Quotient and B ← Remainder)	--	1	4
DEC	A	A ← A – 1	Register	1	1
	Rn	Rn ← Rn – 1	Register	1	1
	Direct	(Direct) ← (Direct) – 1	Direct	2	1
	@Ri	@Ri ← @Ri – 1	Indirect	1	1
INC	A	A ← A + 1	Register	1	1
	Rn	Rn ← Rn + 1	Register	1	1
	Direct	(Direct) ← (Direct) + 1	Direct	2	1
	@Ri	@Ri ← @Ri + 1	Indirect	1	1
	DPTR	DPTR ← DPTR + 1	Register	1	2
DA	A	Decimal Adjust Accumulator	--	1	1

- *ADD*
- *ADDC*
- *SUBB*
- *INC*
- *DEC*
- *MUL*
- *DIV*
- *DA A*

The arithmetic instructions has no knowledge about the data format i.e. signed, unsigned, ASCII, BCD, etc. Also, the operations performed by the arithmetic instructions affect flags like carry, overflow, zero, etc. in the PSW Register.

Logical Instructions

The next group of instructions are the Logical Instructions, which perform logical operations like AND, OR, XOR, NOT, Rotate, Clear and Swap. Logical Instruction are performed on Bytes of data on a bit-by-bit basis.

Mnemonics associated with Logical Instructions are as follows:

- *ANL*
- *ORL*
- *XRL*
- *CLR*
- *CPL*
- *RL*
- *RLC*
- *RR*
- *RRC*
- *SWAP*

Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
ANL	A, #Data	A ← A AND Data	Immediate	2	1
	A, Rn	A ← A AND Rn	Register	1	1
	A, Direct	A ← A AND (Direct)	Direct	2	1
	A, @Ri	A ← A AND @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) AND A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) AND #Data	Direct	3	2
ORL	A, #Data	A ← A OR Data	Immediate	2	1
	A, Rn	A ← A OR Rn	Register	1	1
	A, Direct	A ← A OR (Direct)	Direct	2	1
	A, @Ri	A ← A OR @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) OR A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) OR #Data	Direct	3	2
XRL	A, #Data	A ← A XRL Data	Immediate	2	1
	A, Rn	A ← A XRL Rn	Register	1	1
	A, Direct	A ← A XRL (Direct)	Direct	2	1
	A, @Ri	A ← A XRL @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) XRL A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) XRL #Data	Direct	3	2
CLR	A	A← 00H	--	1	1
CPL	A	A ← A	--	1	1
RL	A	Rotate ACC Left	--	1	1
RLC	A	Rotate ACC Left through Carry	--	1	1
RR	A	Rotate ACC Right	--	1	1
RRC	A	Rotate ACC Right through Carry	--	1	1
SWAP	A	Swap Nibbles within ACC	--	1	1

Boolean or Bit Manipulation Instructions

As the name suggests, Boolean or Bit Manipulation Instructions will deal with bit variables. We know that there is a special bit-addressable area in the RAM and some of the Special Function Registers (SFRs) are also bit addressable.

The Mnemonics corresponding to the Boolean or Bit Manipulation instructions are:

- *CLR*
- *SETB*
- *MOV*
- *JC*
- *JNC*
- *JB*
- *JNB*
- *JBC*
- *ANL*
- *ORL*
- *CP*

Mnemonic	Instruction	Description	# of Bytes	# of Cycles
CLR	C	C ← 0 (C = Carry Bit)	1	1
	Bit	Bit ← 0 (Bit = Direct Bit)	2	1
SET	C	C ← 1	1	1
	Bit	Bit ← 1	2	1
CPL	C	C ← $\overline{C}$	1	1
	Bit	Bit ← $\overline{Bit}$	2	1
ANL	C, /Bit	C ← C. $\overline{Bit}$ (AND)	2	1
	C, Bit	C ← C . Bit (AND)	2	1
ORL	C, /Bit	C ← C + $\overline{Bit}$ (OR)	2	1
	C, Bit	C ← C + Bit (OR)	2	1
MOV	C, Bit	C ← Bit	2	1
	Bit, C	Bit ← C	2	2
JC	rel	Jump is Carry (C) is Set	2	2
JNC	rel	Jump is Carry (C) is Not Set	2	2
JB	Bit, rel	Jump is Direct Bit is Set	3	2
JNB	Bit, rel	Jump is Direct Bit is Not Set	3	2
JBC	Bit, rel	Jump is Direct Bit is Set and Clear Bit	3	2

Program Branching Instructions

The last group of instructions in the 8051 Microcontroller Instruction Set are the Program Branching Instructions. These instructions control the flow of program logic. The mnemonics are as follows. All these instructions, except the NOP (No Operation) affect the Program Counter (PC) in one way or other.

Mnemonic	Instruction	Description	# of Bytes	# of Cycles
ACALL	ADDR11	Absolute Subroutine Call PC + 2 → (SP); ADDR11 → PC	2	2
LCALL	ADDR16	Long Subroutine Call PC + 3 → (SP); ADDR16 → PC	3	2
RET	--	Return from Subroutine (SP) → PC	1	2
RETI	--	Return from Interrupt	1	2
AJMP	ADDR11	Absolute Jump ADDR11 → PC	2	2
LJMP	ADDR16	Long Jump ADDR16 → PC	3	2
SJMP	rel	Short Jump PC + 2 + rel → PC	2	2
JMP	@A + DPTR	A + DPTR → PC	1	2
JZ	rel	If A=0, Jump to PC + rel	2	2
JNZ	rel	If A ≠ 0, Jump to PC + rel		
CJNE	A, Direct, rel	Compare (Direct) with A. Jump to PC + rel if not equal	3	2
	A, #Data, rel	Compare #Data with A. Jump to PC + rel if not equal	3	2
	Rn, #Data, rel	Compare #Data with Rn. Jump to PC + rel if not equal	3	2
	@Ri, #Data, rel	Compare #Data with @Ri. Jump to PC + rel if not equal	3	2
DJNZ	Rn, rel	Decrement Rn. Jump to PC + rel if not zero	2	2
	Direct, rel	Decrement (Direct). Jump to PC + rel if not zero	3	2
NOP		No Operation	1	1

- *LJMP*
- *AJMP*
- *SJMP*
- *JZ*
- *JNZ*
- *CJNE*
- *DJNZ*
- *NOP*
- *LCALL*
- *ACALL*
- *RET*
- *RETI*
- *JMP*

ASSEMBLY LANGUAGE PROGRAMMING

Program to add two 8-bit numbers using 8051.

The register A(Accumulator) is used as one operand in the operations. There are seven registers R0 – R7 in different register banks. We can use any of them as the second operand.

We are taking two number 5FH and D8H at location 20H and 21H, After adding them, the result will be stored at location 30H and 31H.

Address	Value
20H	5FH
21H	D8H
	. . .
30H	00H

Address	Value
31H	00H

```
MOVR0,#20H          ;set source address 20H to R0

MOVR1,#30H          ;set destination address 30H to R1

MOVA,@R0            ; take the value from source to register A

MOVR5,A             ; Move the value from A to R5

MOVR4,#00H          ; Clear register R4 to store carry

INCR0               ; Point to the next location

MOVA,@R0            ; take the value from source to register A

ADDA,R5             ;Add R5 with A and store to register A

JNC SAVE

INCR4               ; Increment R4 to get carry

MOVB,R4             ;Get carry to register B

MOV@R1,B            ; Store the carry first

INCR1               ; Increase R1 to point to the next address

SAVE:   MOV@R1,A;Store the result

HALT:   SJMP HALT ;Stop the program
```

Output

Address	Value

Address	Value
20H	5FH
21H	D8H
	. . .
30H	01H
31H	37H