EINSTEIN COLLEGE OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

EE 64-MICROPROCESSOR AND MICROCONTROLLER

LECTURE NOTES

UNIT I

8085 MICROPROCESSOR

1.1 Hardware Architecture

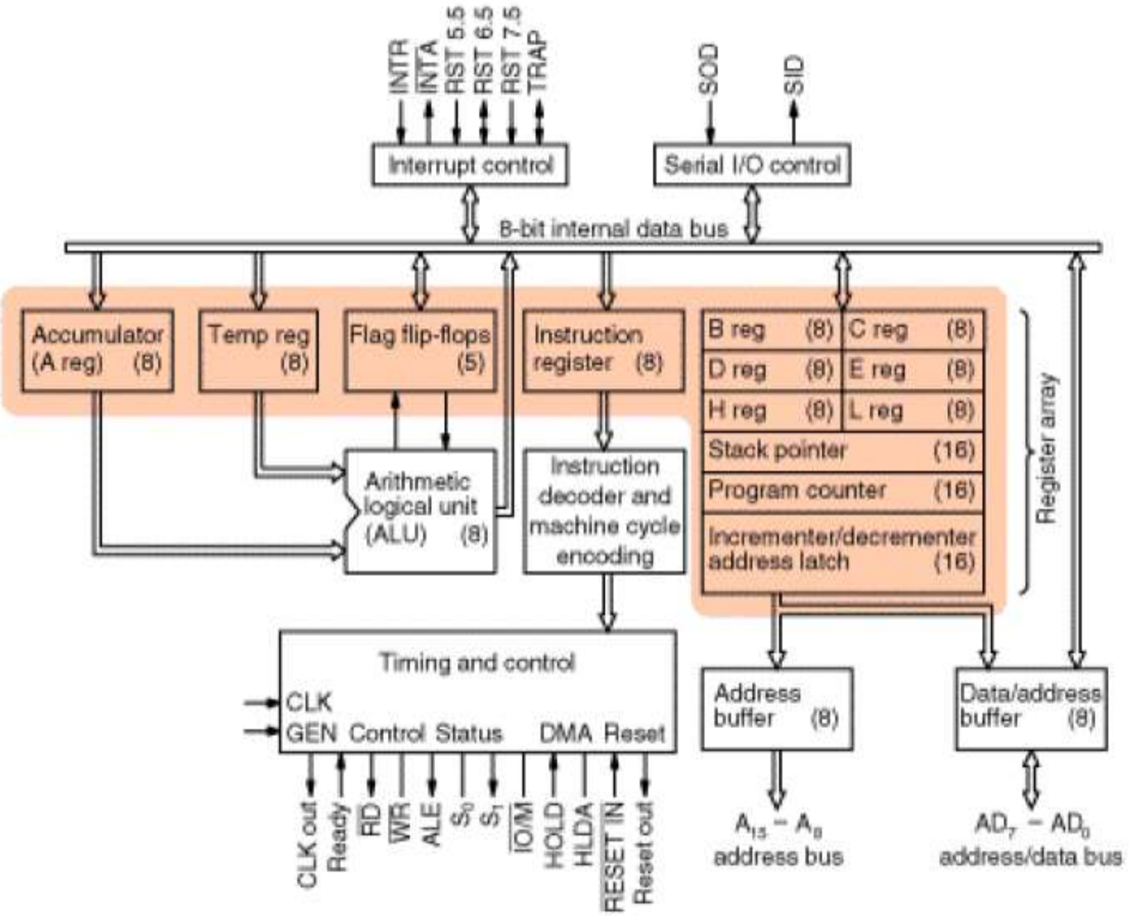


Fig 1.1 Hardware Architecture of 8085

Control Unit

Generates signals within Microprocessor to carry out the instruction, which has been decoded. In reality causes certain connections between blocks of the uP to be opened or closed, so that data goes where it is required, and so that ALU operations occur.

Arithmetic Logic Unit

The ALU performs the actual numerical and logic operation such as ‘add’, ‘subtract’, ‘AND’, ‘OR’, etc. Uses data from memory and from Accumulator to perform arithmetic. Always stores result of operation in Accumulator.

Registers

The 8085/8080A-programming model includes six registers, one accumulator, and one flag register, as shown in Figure. In addition, it has two 16-bit registers: the stack pointer and the program counter. The 8085/8080A has six general-purpose registers to store 8-bit data; these are identified as B,C,D,E,H, and L as shown in the figure. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions.

Accumulator

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

Flags

The ALU includes five flip-flops, which are set or reset after an operation according to 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. The most commonly used flagsare Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.

For example, after an addition of two numbers, if the sum in the accumulator id larger than eight bits, the flip-flop uses to indicate a carry -- called the Carry flag (CY) – is set to one. When an arithmetic operation results in zero, the flip-flop called the Zero(Z) flag is set to one. The first Figure shows an 8-bit register, called the flag register, adjacent to the accumulator. However, it is not used as a register; five bit positions out of eight are used to store the outputs of the five flip-flops. The flags are stored in the 8-bit register so that the programmer can examiexamine these flags (dataconditions) 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 example, the instruction JC (Jump on Carry) is implemented to change the sequence of a program when CY flag is set.

Program Counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a16-bit register.

The microprocessor uses this register to sequence the execution of the instructions.The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location

Stack Pointer (SP)

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

Instruction Register/Decoder

Temporary store for the current instruction of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and ‘decodes’ or interprets the instruction. Decoded instruction then passed to next stage.

Memory Address Register

Holds address, received from PC, of next program instruction. Feeds the address bus with addresses of location of the program under execution.

Control Generator

Generates signals within uP to carry out the instruction which has been decoded. In reality causes certain connections between blocks of the uP to be opened or closed, sothat data goes where it is required, and so that ALU operations occur.

Register Selector

This block controls the use of the register stack in the example. Just a logic circuit which switches between different registers in the set will receive instructions from Control Unit.

1.2 Pin Diagram

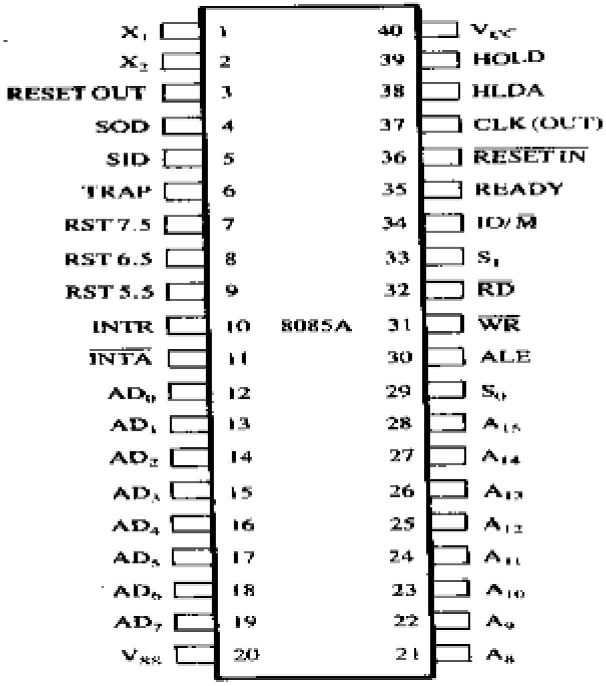


Fig 1.2 Pin Diagram of 8085

A8 - A15 (Output 3 State)

Address Bus:The most significant 8 bits of the memory address or the 8 bits of the I/0 address,3 stated during Hold and Halt modes.

AD0 - AD7 (Input/Output 3state)

Multiplexed Address/Data Bus; Lower 8 bits of the memory address (or I/0 address) appear on the bus during the first clock cycle of a machine state. It then becomes thedata bus during the second and third clock cycles. 3 stated during Hold and Halt modes.

ALE (Output)

Address Latch Enable: It occurs during the first clock cycle of a machine state and enables the address to get latched into the on chip latch of peripherals. The falling edge of ALE is set to guarantee setup and hold times for the address information. ALE can also be used to strobe the status information. ALE is never 3stated.

SO, S1 (Output)

Data Bus Status. Encoded status of the bus cycle:

S1	S0	
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

S1 can be used as an advanced R/W status.

RD (Output 3state)

READ: indicates the selected memory or 1/0 device is to be read and that the Data Bus is available for the data transfer.

WR (Output 3state)

WRITE:indicates the data on the Data Bus is to be written into the selected memory or 1/0 location. Data is set up at the trailing edge of WR. 3stated during Hold and Halt modes.

READY (Input)

If Ready is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If Ready is low, the CPU will wait forReady to go high before completing the read or write cycle.

HOLD (Input)

HOLD:indicates that another Master is requesting the use of the Address and DataBuses. The CPU, upon receiving the Hold request. will relinquish the use of buses as soon as the completion of the current machine cycle. Internal processing can continue.The processorcanregain the buses only after the Hold is removed. When the Hold is acknowledged, the Address, Data, RD, WR, and IO/M lines are 3stated.

HLDA (Output)

HOLD ACKNOWLEDGE:indicates that the CPU has received the Hold request and that it will relinquish the buses in the next clock cycle. HLDA goes low after the Hold request is removed. The CPU takes the buses one half clock cycle after HLDA goes low.

INTR (Input)

INTERRUPT REQUEST is used as a general purpose interrupt. It is sampled onlyduring the next to the last clock cycle of the instruction. If it is active, the Program Counter (PC) will be inhibited from incrementing and an INTA will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.

INTA (Output)

INTERRUPT ACKNOWLEDGE: is used instead of (and has the same timing as) RDduring the Instruction cycle after an INTR is accepted. It can be used to activate the 8259 Interrupt chip or some other interrupt port.

RESTART INTERRUPTS

These three inputs have the same timing as INTR except they cause an internal RESTART to be automatically inserted.

RST 7.5 ~~ Highest Priority

RST 6.5

RST 5.5 Lowest Priority

TRAP (Input)

Trap interrupt is a nonmaskable restart interrupt. It is recognized at the same time asINTR. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt.

RESET IN (Input)

Reset sets the Program Counter to zero and resets the Interrupt Enable and HLDA flipflops. None of the other flags or registers (except the instruction register) are affected The CPU is held in the reset condition as long as Reset is applied.

RESET OUT (Output)

Indicates CPlJ is being reset. Can be used as a system RESET. The signal is synchronized to the processor clock.

X1, X2 (Input)

Crystal or R/C network connections to set the internal clock generator X1 can also be an external clock input instead of a crystal. The input frequency is divided by 2 togive the internal operating frequency.

CLK (Output)

Clock Output for use as a system clock when a crystal or R/ C network is used as an input to the CPU. The period of CLK is twice the X1, X2 input period.

IO/M (Output)

IO/M indicates whether the Read/Write is to memory or l/O Tristated during Hold and Halt modes.

SID (Input)

Serial input data line The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

SOD (output)

Serial output data line. The output SOD is set or reset as specified by the SIM instruction.

Vcc

+5 volt supply.

Vss

Ground Reference.

1.3 Memory Interfacing

The memory is made up of semiconductor material used to store the programs and data. Three types of memory is,

- *Process memory*
- *Primary or main memory*
- *Secondary memory*

1.3.1 Typical EPROM and Static RAM

A typical semiconductor memory IC will have n address pins, m data pins (or output pins).

- Having two power supply pins (one for connecting required supply voltage (V and the other for connecting ground).
- The control signals needed for static RAM are chip select (chip enable), read control (output enable) and write control (write enable).
- The control signals needed for read operation in EPROM are chip select (chip enable) and read control (output enable).

1.3.2 **Decoder**

It is used to select the memory chip of processor during the execution of a program. No of IC's used for decoder is,

- 2-4 decoder (74LS139)
- 3-8 decoder (74LS138*)*

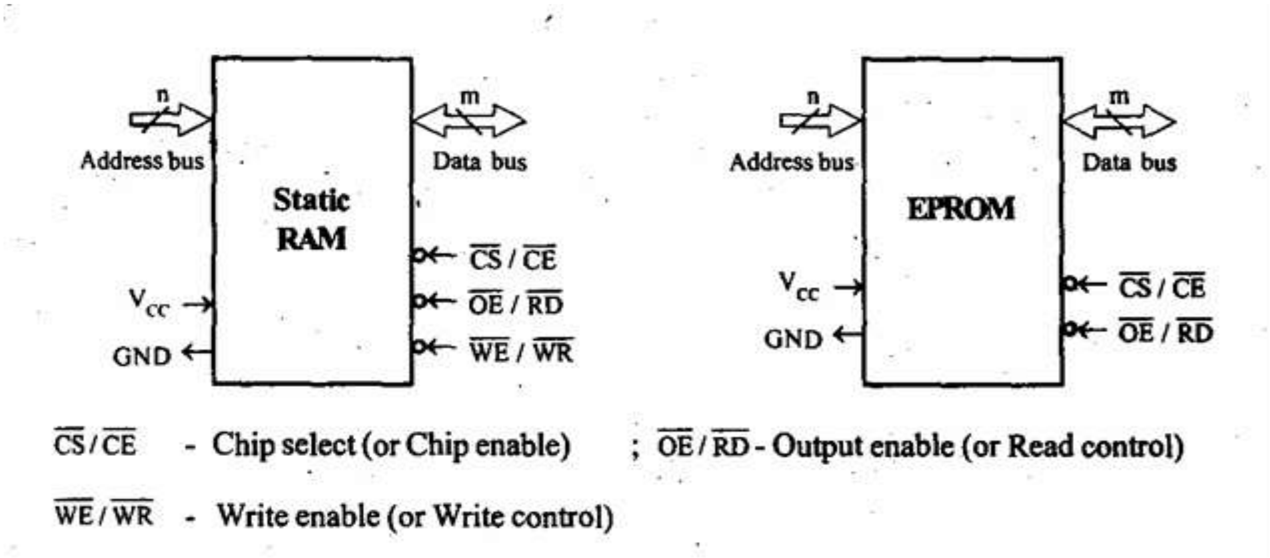
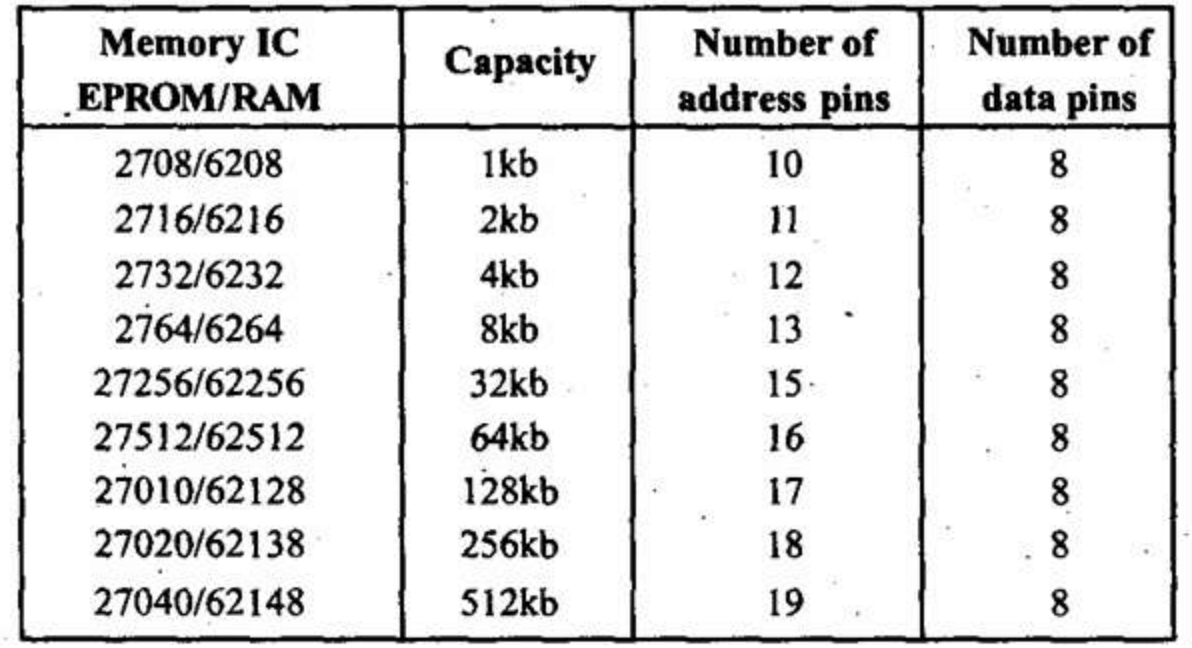

n
Address bus
Static RAM
m
Data bus
$\overline{CS}$ / $\overline{CE}$
$\overline{OE}$ / $\overline{RD}$
$\overline{WE}$ / $\overline{WR}$
V_{cc}
GND
n
Address bus
EPROM
m
Data bus
$\overline{CS}$ / $\overline{CE}$
$\overline{OE}$ / $\overline{RD}$
V_{cc}
GND
$\overline{CS}$ / $\overline{CE}$ - Chip select (or Chip enable) ; $\overline{OE}$ / $\overline{RD}$ - Output enable (or Read control)
$\overline{WE}$ / $\overline{WR}$ - Write enable (or Write control)

Fig 1.3 Static RAM and EPROM

Table1.1 Number of Address Pins and Data Pins in Memory ICs



Memory IC EPROM/RAM	Capacity	Number of address pins	Number of data pins
2708/6208	1kb	10	8
2716/6216	2kb	11	8
2732/6232	4kb	12	8
2764/6264	8kb	13	8
27256/62256	32kb	15	8
27512/62512	64kb	16	8
27010/62128	128kb	17	8
27020/62138	256kb	18	8
27040/62148	512kb	19	8

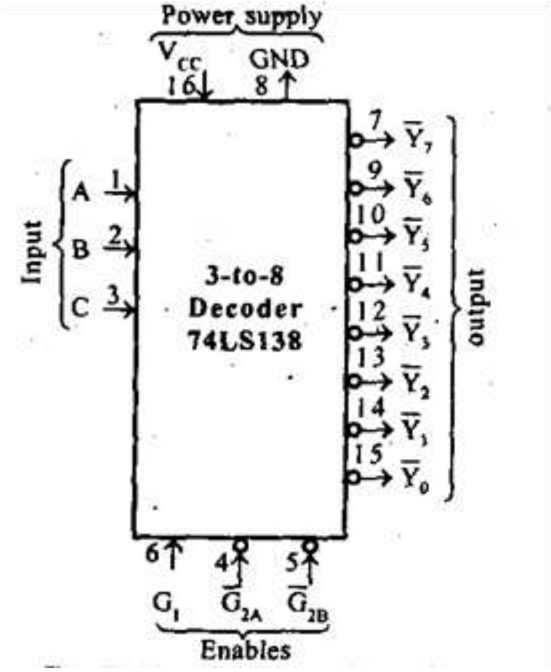
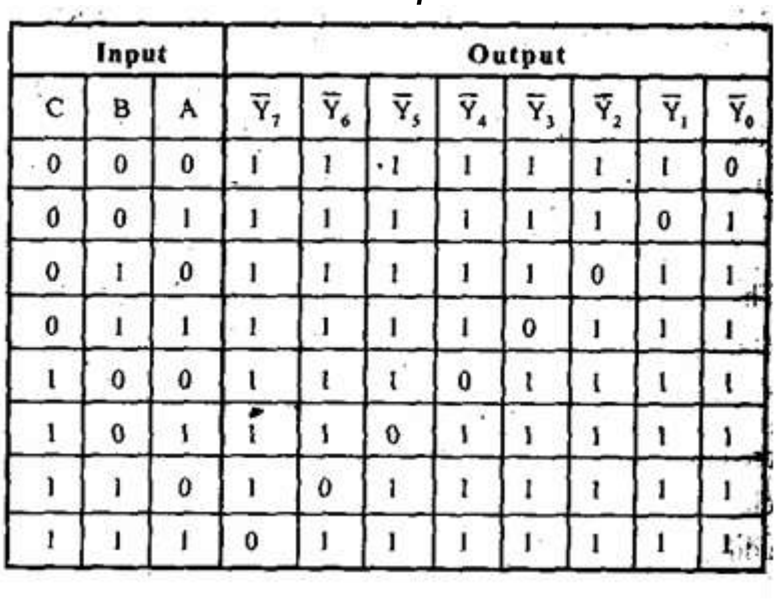


Fig 1.4 Block Diagram of 3 to 8 Decoder

Table 1.2 Truth Table for 3 to 8 decoder



Input			Output							
C	B	A	$\overline{Y}_7$	$\overline{Y}_6$	$\overline{Y}_5$	$\overline{Y}_4$	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	1	1	0	1	1	1	1
1	0	1	1	1	0	1	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1

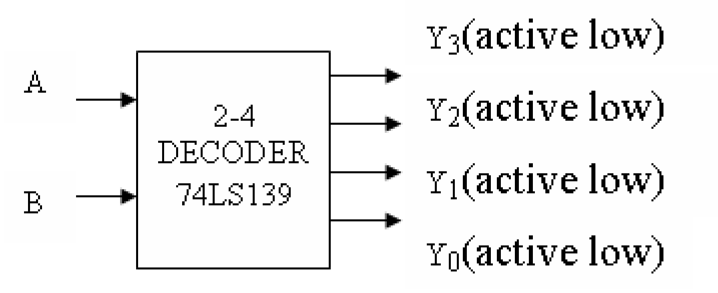
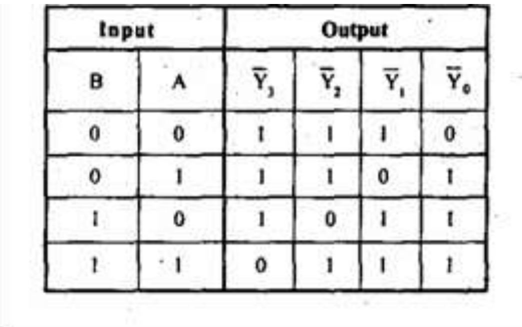


Fig 1.5 Block Diagram of 2-4 Decoder

Table 1.3 Truth Table for 2-4 Decoder



Input		Output			
B	A	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

1.3.3 Example for Memory Interfacing

Consider a system in which the full memory space 64kb is utilized for EPROM memory. Interface the EPROM with 8085 processor.

- The memory capacity is 64 Kbytes. i.e

- 2^n = 64 x 1000 bytes where n = address lines.
- So, n = 16.
- In this system the entire 16 address lines of the processor are connected to address input pins of memory IC in order to address the internal locations of memory.
- The chip select (CS) pin of EPROM is permanently tied to logic low (i.e., tied to ground).
- Since the processor is connected to EPROM, the active low RD pin is connected to active low output enable pin of EPROM.
- The range of address for EPROM is 0000H to FFFFH.

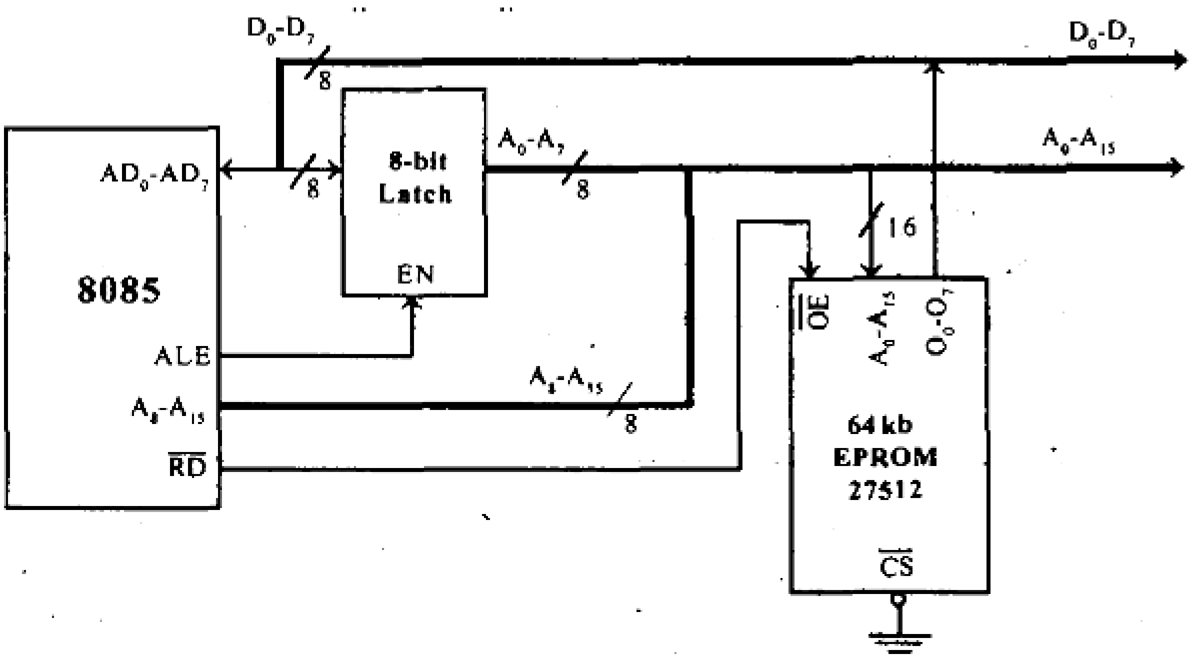


Fig 1.6 Memory Interfacing

1.4 Timing Diagram

Timing Diagram is a graphical representation. It represents the execution time taken by each instruction in a graphical format. The execution time is represented in T-states.

- **Instruction Cycle**

The time required to execute an instruction is called instruction cycle.

- **Machine Cycle**

  The time required to access the memory or input/output devices is called machine cycle.

- **T-State**

  ❖ The machine cycle and instruction cycle takes multiple clock periods.

  ❖ A portion of an operation carried out in one system clock period is called as T-state.

1.4.1 Machine cycles of 8085

The 8085 microprocessor has 5 (seven) basic machine cycles. They are

1. Opcode fetch cycle (4T)
2. Memory read cycle (3 T)
3. Memory write cycle (3 T)
4. I/O read cycle (3 T)
5. I/O write cycle (3 T)

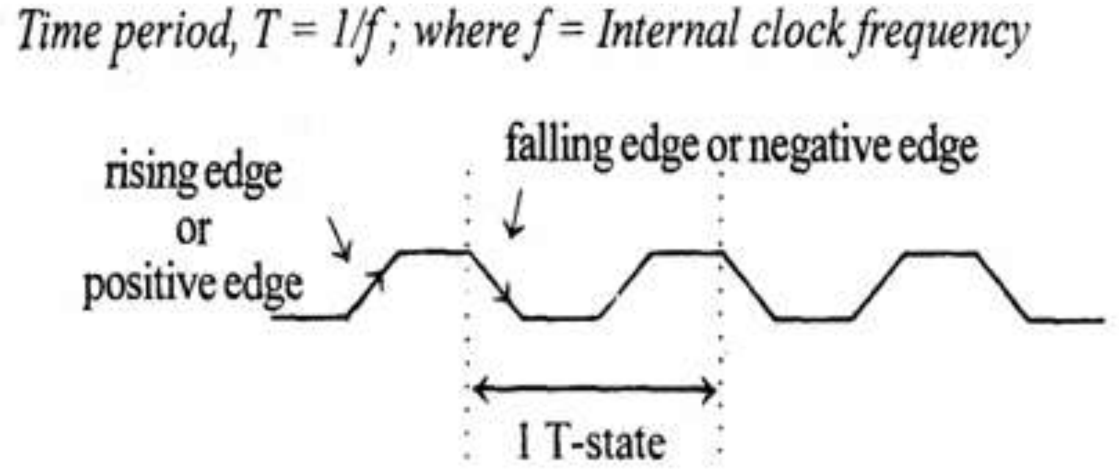

Time period, T = 1/f ; where f = Internal clock frequency
rising edge
or
positive edge
falling edge or negative edge
1 T-state

Fig 1.7 Clock Signal

1.Opcode fetch machine cycle of 8085 :

- Each instruction of the processor has one byte opcode.

- The opcodes are stored in memory. So, the processor executes the opcode fetch machine cycle to fetch 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.
- 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.

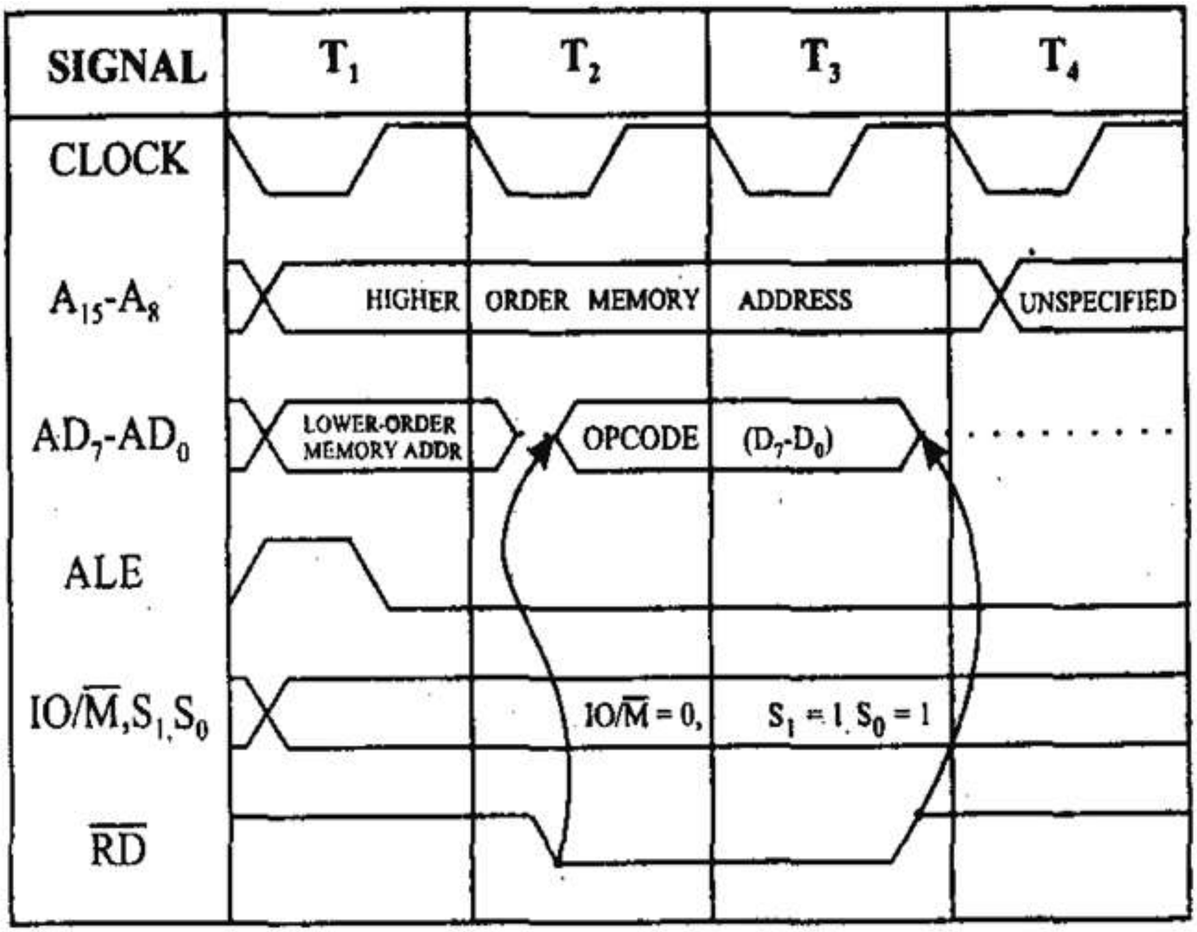


Fig 1.8 Opcode fetch machine cycle

2.Memory Read Machine Cycle of 8085:

- 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.

The instructions which have more than one byte word size will use the machine cycle after the opcode fetch machine cycle.

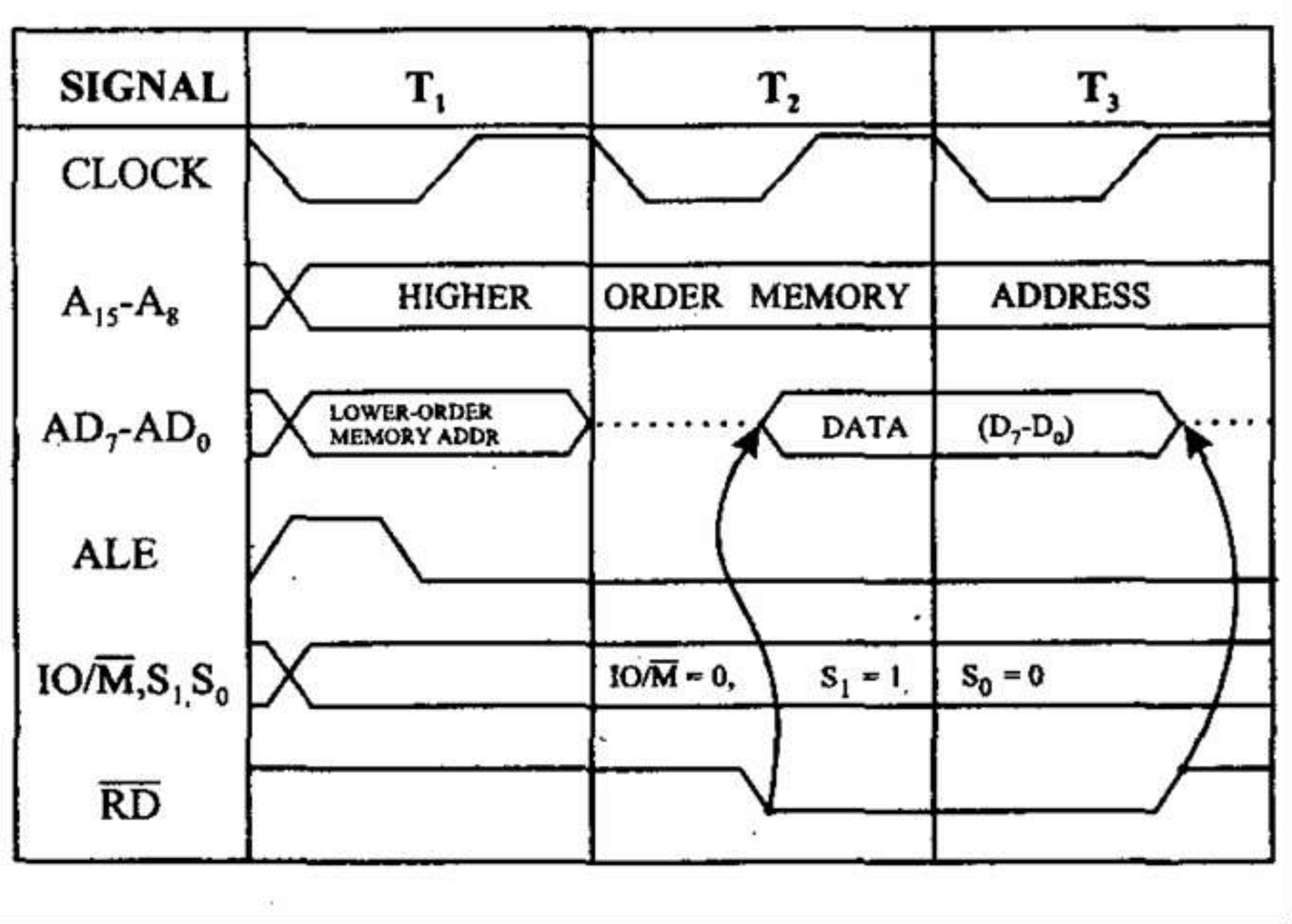


Fig 1.9 Memory Read Machine Cycle

3.Memory Write Machine Cycle of 8085

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

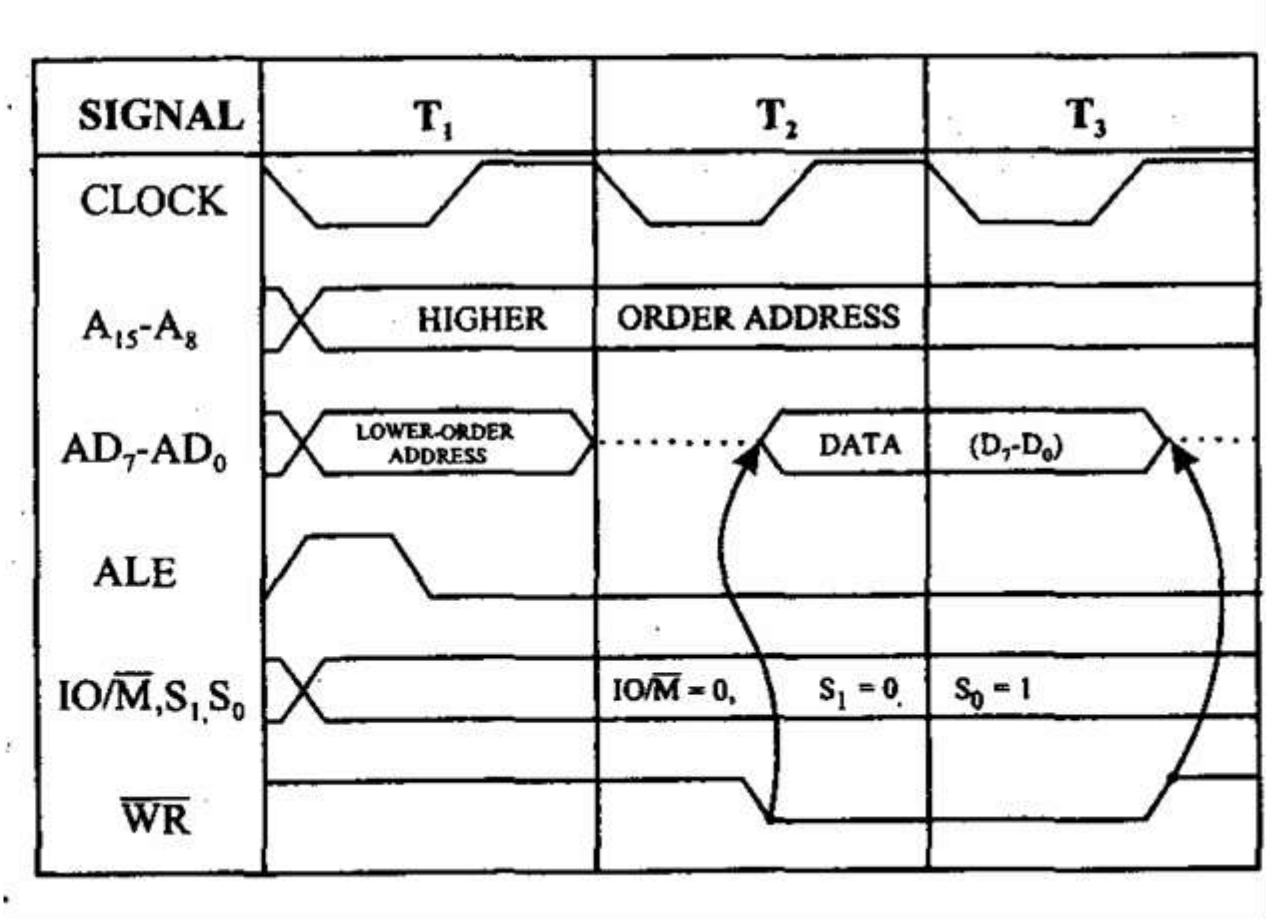


Fig 1.10 Memory Write Machine Cycle

4. I/O Read Cycle of 8085

- The I/O Read cycle is executed by the processor to read a data byte from I/O port or from the peripheral, which is I/O, mapped in the system.
- The processor takes 3T states to execute this machine cycle.
- The IN instruction uses this machine cycle during the execution.

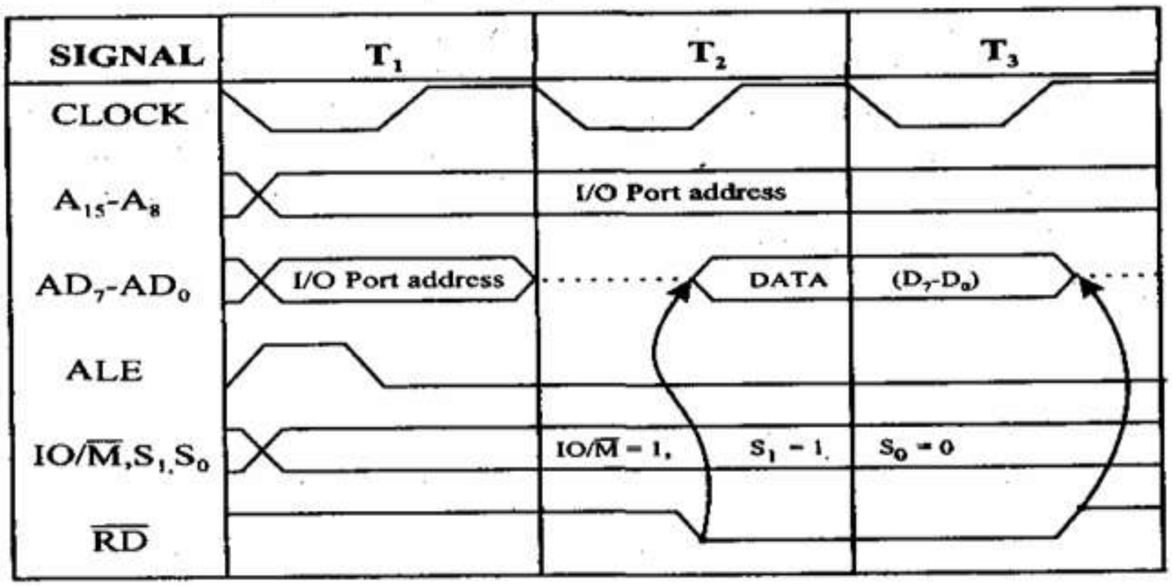


Fig 1.11 I/O Read Cycle

1.4.2 Timing diagram for STA 526AH

- STA means Store Accumulator -The contents of the accumulator is stored in the specified address(526A).
- The opcode of the STA instruction is said to be 32H. It is fetched from the memory 41FFH(see fig). - OF machine cycle
- Then the lower order memory address is read(6A). - Memory Read Machine Cycle
- Read the higher order memory address (52).- Memory Read Machine Cycle
- The combination of both the addresses are considered and the content from accumulator is written in 526A. - Memory Write Machine Cycle
- Assume the memory address for the instruction and let the content of accumulator is C7H. So, C7H from accumulator is now stored in 526A.

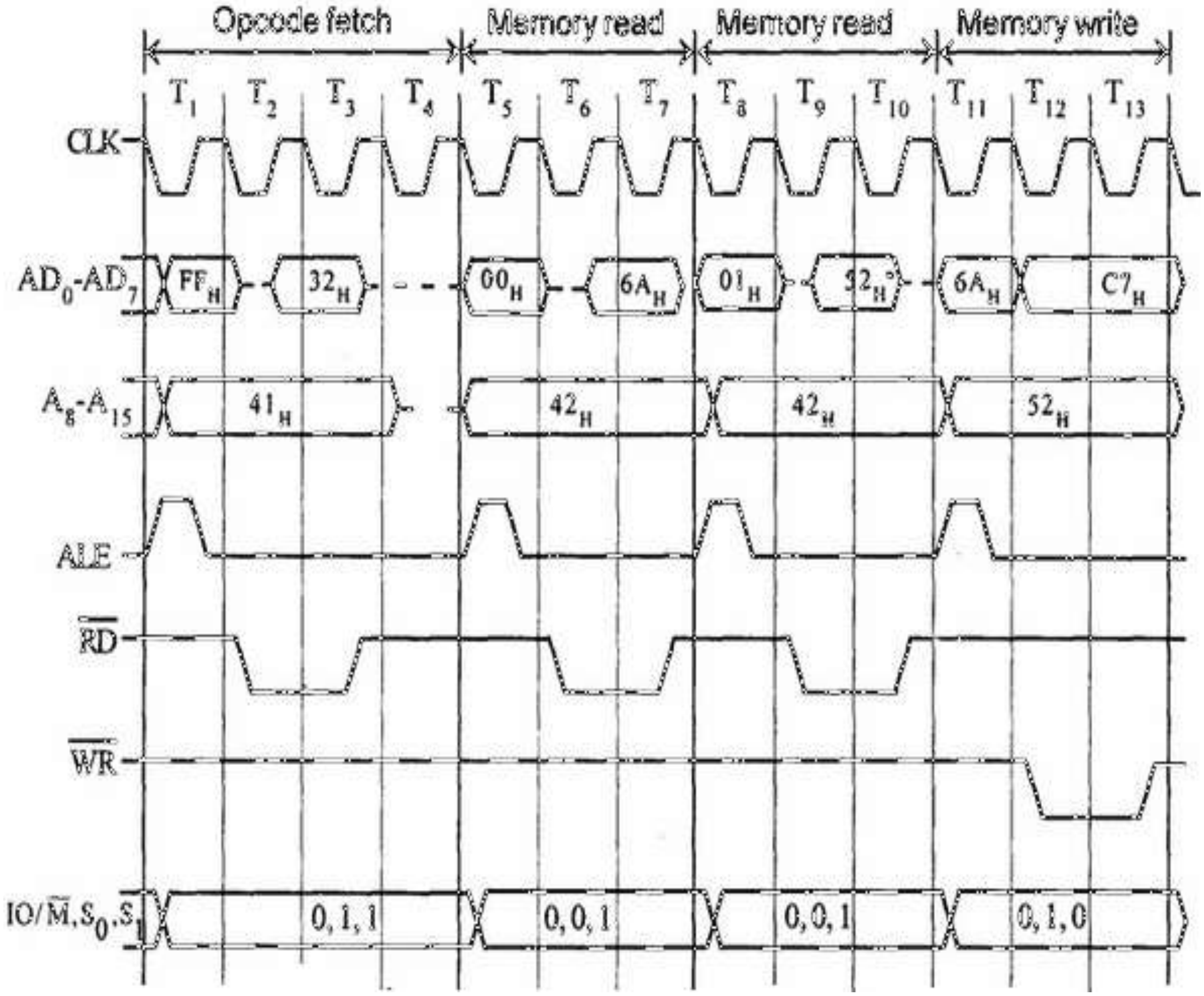
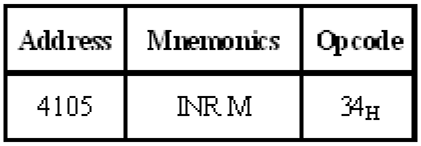


Fig 1.12 Timing Diagram for STA 526A H

1.4.3 Timing diagram for INR M

- Fetching the Opcode 34H from the memory 4105H. (OF cycle)
- Let the memory address (M) be 4250H. (MR cycle -To read Memory address and data)
- Let the content of that memory is 12H.
- Increment the memory content from 12H to 13H. (MW machine cycle)



Address	Mnemonics	Opcode
4105	INR M	34_H

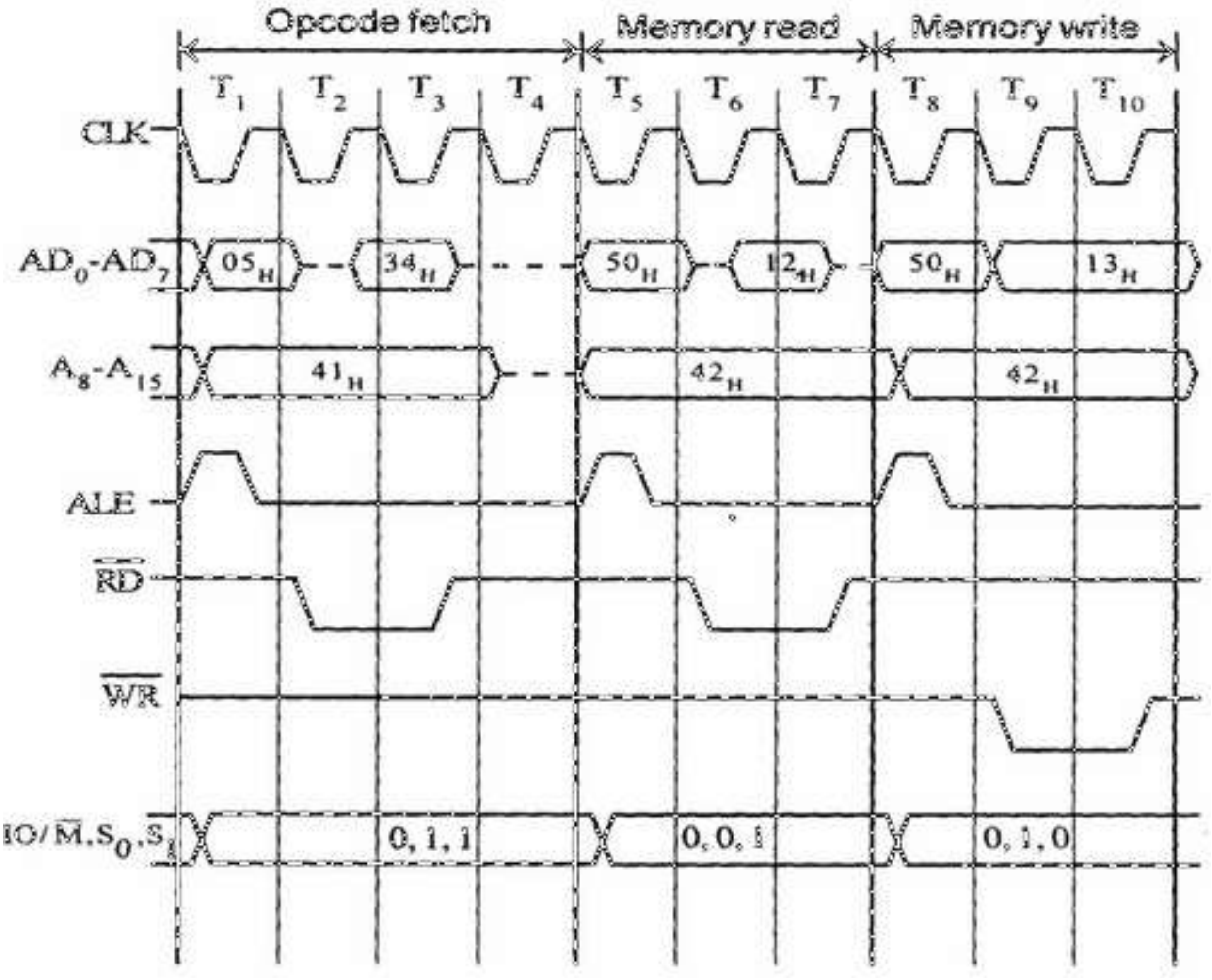
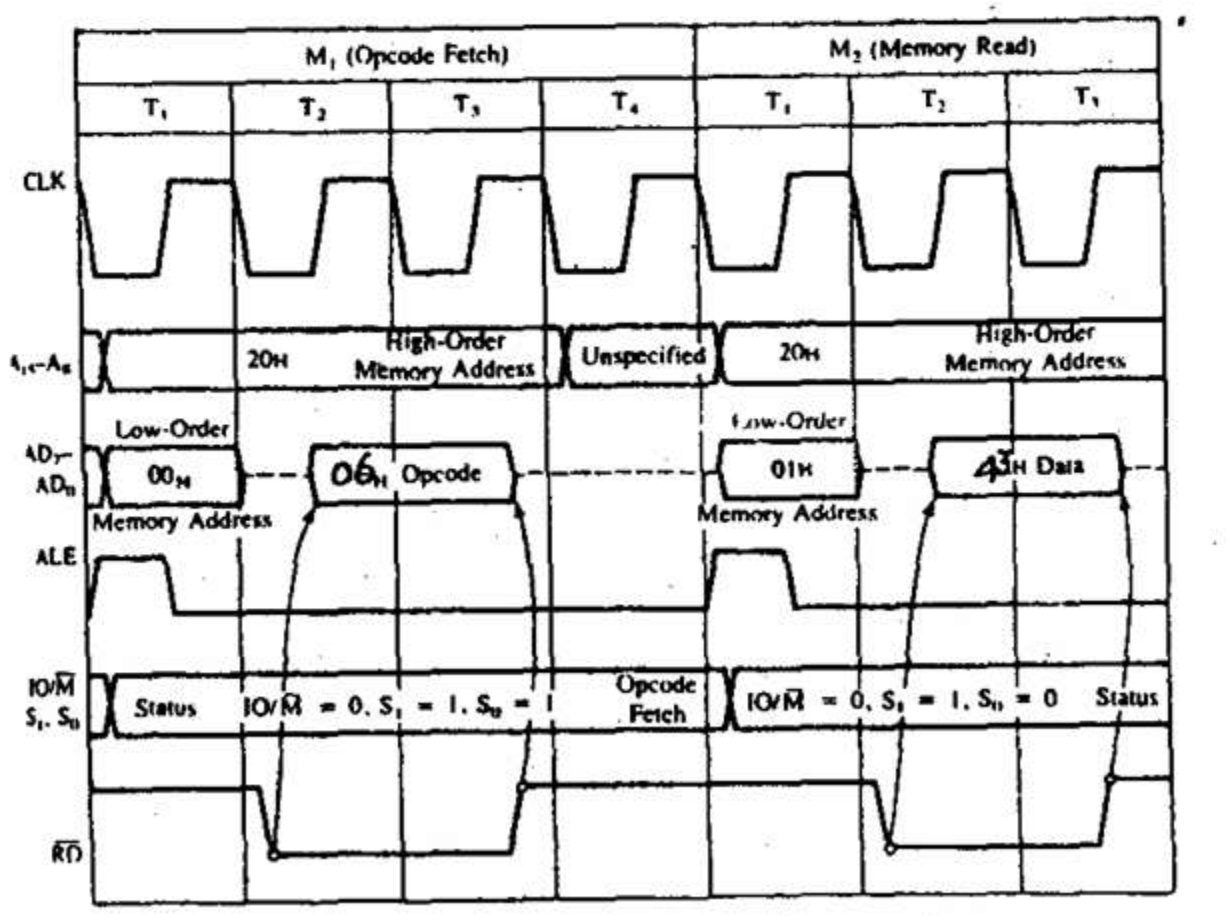


Fig 1.13 Timing Diagram for INR M

1.4.4 Timing diagram for MVI B, 43H.

- Fetching the Opcode 06H from the memory 2000H. (OF machine cycle)
- Read (move) the data 43H from memory 2001H. (memory read)



Address	Mnemonics	Opcode
2000	MVI B, 43_H	06_H
2001		43_H

Fig 1.14 Timing Diagram for MVI B,43 H

1.5 Interrupts:

- Interrupt is signals send by an external device to the processor, to request the processor to perform a particular task or work.

- Mainly in the microprocessor based system the interrupts are used for data transfer between the peripheral and the microprocessor.

- The processor will check the interrupts always at the 2nd T-state of last machine cycle.

- If there is any interrupt it accept the interrupt and send the INTA (active low) signal to the peripheral.

- The vectored address of particular interrupt is stored in program counter.

- The processor executes an interrupt service routine (ISR) addressed in program counter.

- It returned to main program by RET instruction.

1.5.1Types of Interrupts:

It supports two types of interrupts.

- *Hardware*
- *Software*

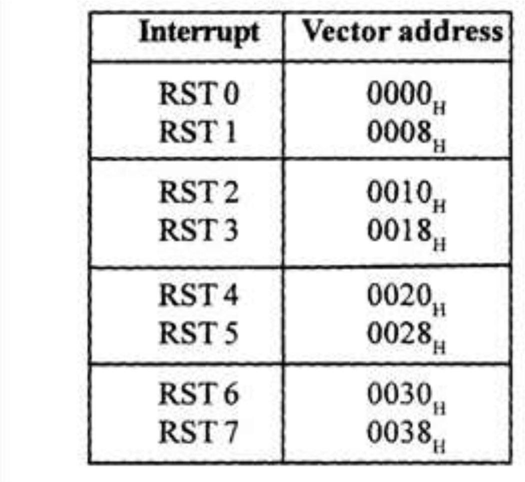
1.5.1.1 Software interrupts:

- *The software interrupts are program instructions. These instructions are inserted at desired locations in a program.*

- *The 8085 has eight software interrupts from RST 0 to RST 7. The vector address for these interrupts can be calculated as follows.*

- *Interrupt number * 8 = vector address*
- *For RST 5,5 * 8 = 40 = 28H*
- *Vector address for interrupt RST 5 is 0028H*

Table 1.4 Vector addresses of all interrupts.



Interrupt	Vector 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

1.5.1.2 Hardware interrupts:

- An external device initiates the hardware interrupts and placing an appropriate signal at the interrupt pin of the processor.
- If the interrupt is accepted then the processor executes an interrupt service routine.

The 8085 has five hardware interrupts

(1) TRAP *(2) RST 7.5* *(3) RST 6.5* *(4) RST 5.5* *(5) INTR*

(1)TRAP:

- This interrupt is a non-maskable interrupt. It is unaffected by any mask or interrupt enable.
- TRAP bas the highest priority and vectored interrupt.
- TRAP interrupt is edge and level triggered. This means hat the TRAP must go high and remain high until it is acknowledged.

- In sudden power failure, it executes a ISR and send the data from main memory to backup memory.

- The signal, which overrides the TRAP, is HOLD signal. (i.e., If the processor receives HOLD and TRAP at the same time then HOLD is recognized first and then TRAP is recognized).

- There are two ways to clear TRAP interrupt.

  1.By resetting microprocessor (External signal)

  2.By giving a high TRAP ACKNOWLEDGE (Internal signal)

(2)RST 7.5:

- The RST 7.5 interrupt is a maskable interrupt.

- It has the second highest priority.

- It is edge sensitive. ie. Input goes to high and no need to maintain high state until it recognized.

- Maskable interrupt. It is disabled by,

  1.DI instruction
  2.System or processor reset.
  3.After reorganization of interrupt.

- Enabled by EI instruction**.**

(3)RST 6.5 and 5.5:

- The RST 6.5 and RST 5.5 both are level triggered. . ie. Input goes to high and stay high until it recognized.

- Maskable interrupt. It is disabled by,

  1.DI, SIM instruction

  2.System or processor reset.

3.After reorganization of interrupt.

- Enabled by EI instruction.
- The RST 6.5 has the third priority whereas RST 5.5 has the fourth priority.

(4)INTR:

- INTR is a maskable interrupt. It is disabled by,

  1.DI, SIM instruction
  2.System or processor reset.
  3.After reorganization of interrupt

- Enabled by EI instruction.
- Non- vectored interrupt. After receiving INTA (active low) signal, it has to supply the address of ISR.
- It has lowest priority.
- It is a level sensitive interrupts. ie. Input goes to high and it is necessary to maintain high state until it recognized.

The following sequence of events occurs when INTR signal goes high.

1. The 8085 checks the status of INTR signal during execution of each instruction.

2. If INTR signal is high, then 8085 complete its current instruction and sends active low interrupt acknowledge signal, if the interrupt is enabled.

3. In response to the acknowledge signal, external logic places an instruction OPCODE on the data bus. In the case of multibyte instruction, additional interrupt acknowledge machine cycles are generated by the 8085 to transfer the additional bytes into the microprocessor.

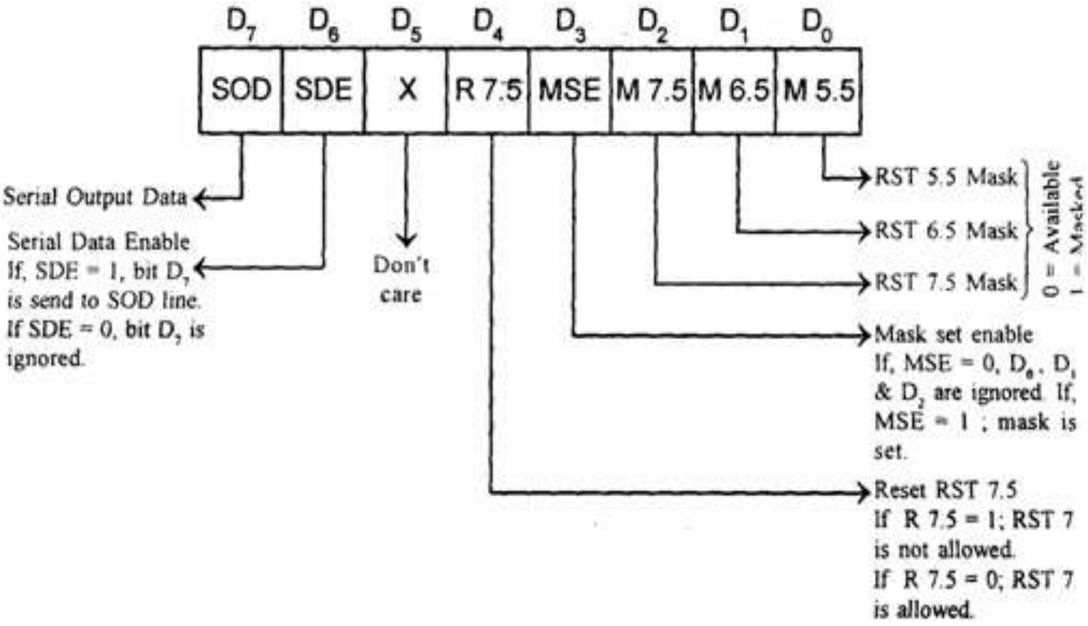
4. On receiving the instruction, the 8085 save the address of next instruction on stack and execute received instruction.

SIM and RIM for interrupts:

- The 8085 provide additional masking facility for RST 7.5, RST 6.5 and RST 5.5 using SIM instruction.

- The status of these interrupts can be read by executing RIM instruction.

- The masking or unmasking of RST 7.5, RST 6.5 and RST 5.5 interrupts can be performed by moving an 8-bit data to accumulator and then executing SIM instruction.



D_7
D_6
D_5
D_4
D_3
D_2
D_1
D_0
SOD
SDE
X
R 7.5
MSE
M 7.5
M 6.5
M 5.5
Serial Output Data
Serial Data Enable If, SDE = 1, bit D_7 is send to SOD line. If SDE = 0, bit D_7 is ignored.
Don't care
RST 5.5 Mask
RST 6.5 Mask
RST 7.5 Mask
0 = Available
1 = Masked
Mask set enable If, MSE = 0, D_0, D_1 & D_2 are ignored. If, MSE = 1 ; mask is set.
Reset RST 7.5 If R 7.5 = 1; RST 7 is not allowed. If R 7.5 = 0; RST 7 is allowed.

Fig 1.15 8 bit data to be loaded into the Accumulator

- The status of pending interrupts can be read from accumulator after executing RIM instruction.

- When RIM instruction is executed an 8-bit data is loaded in accumulator, which can be interpreted as shown in fig.

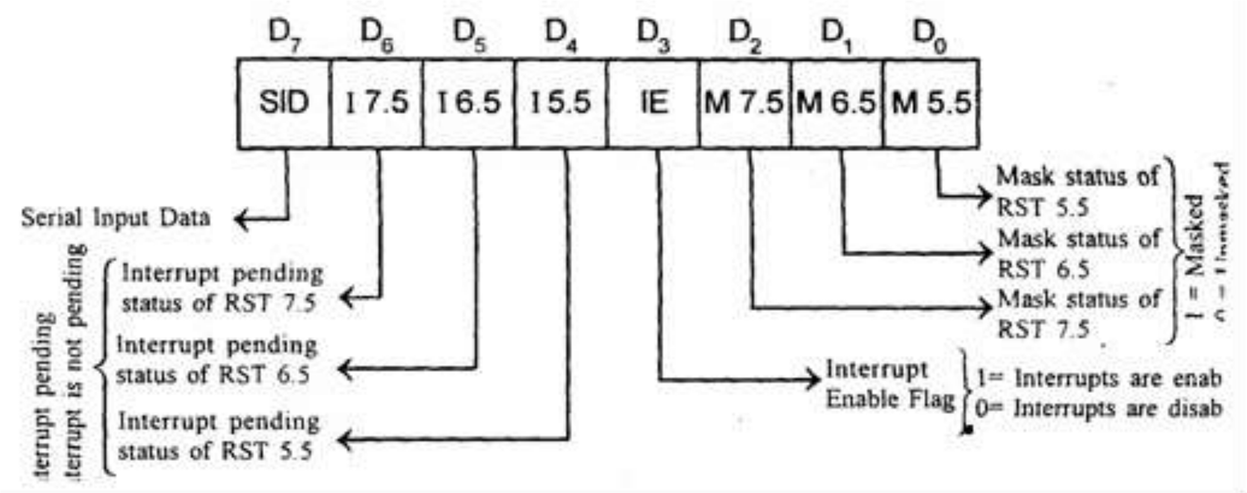


Fig 1.16 Format of 8 bit data in Accumulator after executing RIM Instruction

1.6 Introduction to 8086 Microprocessor

1.6.1 Features:

1. Intel 8086 was launched in 1978.
2. It was the first 16-bit microprocessor.
3. This microprocessor had major improvement over the execution speed of 8085.
4. It is available as 40-pin Dual-Inline-Package (DIP).
5. It is available in three versions:
    a. 8086 (5 MHz)
    b. 8086-2 (8 MHz)
    c. 8086-1 (10 MHz)
6. It consists of 29,000 transistors.

1.6.2 Architecture of 8086 Microprocessor

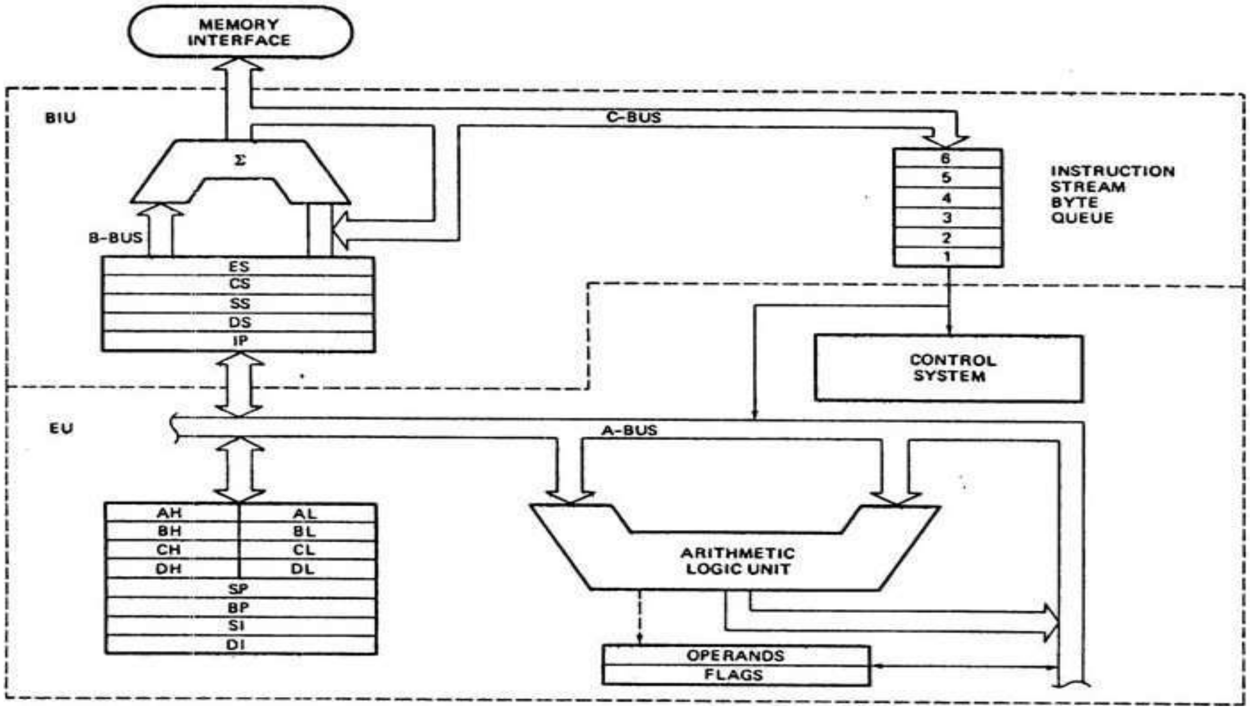


Fig 1.17 Architecture of 8086

1.6.2.1 Bus Interface Unit (BIU):

The function of BIU is to

- Fetch the instruction or data from memory.
- Write the data to memory.
- Write the data to the port.
- Read data from the port.

1.6.2.2 Instruction Queue

1. To increase the execution speed, BIU fetches as many as six instruction bytes ahead to timefrom memory.
2. All six bytes are then held in first in first out 6 byte register called instruction queue.
3. Then all bytes have to be given to EU one by one.
4. This pre fetching operation of BIU may be in parallel with execution operation of

EU, which improves the speed execution of the instruction.

1.6.2.3 Execution Unit (EU)

The functions of execution unit are

- To tell BIU where to fetch the instructions or data from.
- To decode the instructions.
- To execute the instructions.

The EU contains the control circuitry to perform various internal operations. A decoder in EU decodes the instruction fetched memory to generate different internal or external control signals required to perform the operation. EU has 16-bit ALU, which can perform arithmetic and logical operations on 8-bit as well as 16-bit.

1.6.2.4 General Purpose Registers of 8086

These registers can be used as 8-bit registers individually or can be used as 16-bit in pair to have AX,BX, CX, and DX.

1. AX Register: AX register is also known as accumulator register that stores operands for arithmetic operation like divided, rotate.

2. BX Register: This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment.

3. CX Register: It is defined as a counter. It is primarily used in loop instruction to store loop counter.

4.DX Register: DX register is used to contain I/O port address for I/O instruction.

1.6.2.5 Segment Registers

Additional registers called segment registers generate memory address when combined with other in the microprocessor. In 8086 microprocessor, memory is divided into 4 segments as follow:

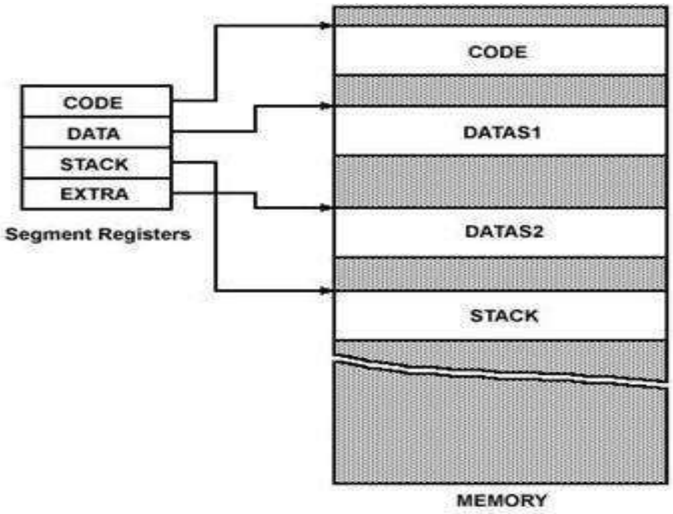


Fig 1.18 Memory Segments of 8086

1. Code Segment (CS): The CS register is used for addressing a memory location in the Code Segment of the memory, where the executable program is stored.

2. Data Segment (DS): The DS contains most data used by program. Data are accessed in the Data Segment by an offset address or the content of other register that holds the offset address.

3. Stack Segment (SS): SS defined the area of memory used for the stack.

4. Extra Segment (ES): ES is additional data segment that is used by some of the string to hold the destination data.

1.6.2.6 Flag Registers of 8086

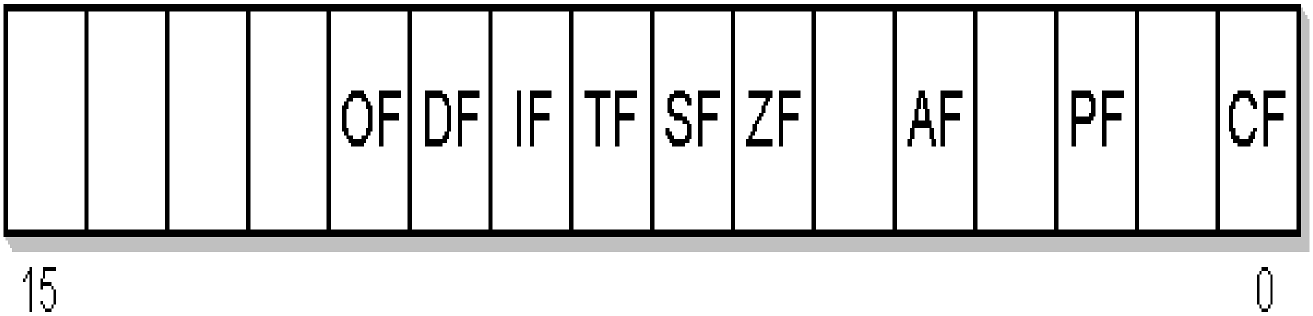


Fig 1.19 Flag Register of 8086

Flags Register determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program. 8086 has 9 flags and they are divided into two categories:

1. Conditional Flags

2. Control Flags

(1) Conditional Flags

Conditional flags represent result of last arithmetic or logical instruction executed. Conditional flags are as follows:

Carry Flag (CF)

This flag indicates an overflow condition for unsigned integer arithmetic.

It is also used in multiple-precision arithmetic.

Auxiliary Flag (AF):

If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. D_0 D_3) to upper nibble (i.e. $D_4 – D_7$), the AF flag is set i.e. carry given by D_3 bit to D_4 is AF flag. This is not a general-purpose flag, it is used internally by the processor to perform Binary to BCD conversion.

Parity Flag (PF):

This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity Flag is reset.

Zero Flag (ZF):

It is set; if the result of arithmetic or logical operation is zero else it is reset.

Sign Flag (SF):

In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.

Overflow Flag (OF):

It occurs when signed numbers are added or subtracted. An OF indicates that the result has exceeded the capacity of machine.

Control Flags

Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows:

1. Trap Flag (TP):

a. *It is used for single step control.*

b. *It allows user to execute one instruction of a program at a time for debugging.*

c. *When trap flag is set, program can be run in single step mode.*

2. Interrupt Flag (IF):

a. *It is an interrupt enable/disable flag.*

b. *If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled.*

c. *It can be set by executing instruction sit and can be cleared by executing CLI instruction.*

3. Direction Flag (DF):

a. *It is used in string operation.*

b. *If it is set, string bytes are accessed from higher memory address to lower memory address.*

c. *When it is reset, the string bytes are accessed from lower memory address to higher memory address.*

1.6.3 8086-Minimum mode of operation

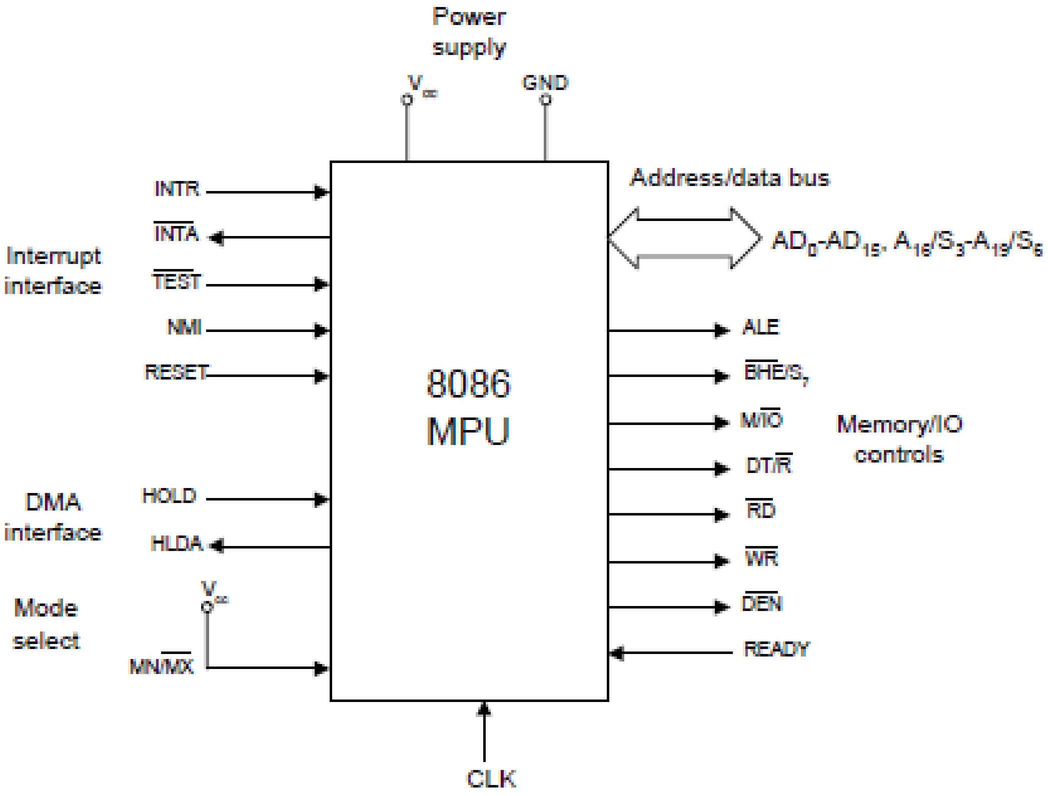


Fig 1.20 Minimum mode of 8086

1.6.3.1Minimum Mode Interface

- **Address/Data bus:** 20 bits vs 8 bits multiplexed
- **Status signals:** A16-A19 multiplexed with status signals S3-S6 respectively

– S3 and S4 together form a 2 bit binary code that identifies which of the internal segment registers was used to generate the physical address that was output on the address bus during the current bus cycle.

– S5 is the logic level of the internal interrupt enable flag, s6 is always logic 0.

- **Control Signals:**

– **Address Latch Enable (ALE)** is a pulse to logic 1 that signals external circuitry when a valid address is on the bus. This address can be latched in external circuitry on the 1-to-0 edge of the pulse at ALE.

– **IO/M line:** memory or I/O transfer is selected (complement for 8086)

– **DT/R line**: direction of data is selected

– **SSO (System Status Output) line**: =1 when data is read from memory and =0 when code is read from memory (only for 8088)

– **BHE (Bank High Enable) line** : =0 for most significant byte of data for
8086 and also carries S7

– **RD line: =**0 when a read cycle is in progress

– **WR line:** =0 when a write cyle is in progress

– **DEN line: (Data enable)** Enables the external devices to supply data
to the processor.

– **Ready line:** can be used to insert wait states into the bus cycle so that it is extended by a number of clock periods

- **Interrupt signals:**

– **INTR (Interrupt request)** :=1 shows there is a service request, sampled at the final clock cycle of each instruction acquisition cycle.

– **INTA** : Processor responds with two pulses going to 0 when it services the interrupt and waits for the interrupt service number after the second pulse.

– **TEST**: Processor suspends operation when =1. Resumes operation when=0. Used to syncronize the processor to external events.

– NMI (Nonmaskable interrupt) : A leading edge transition causes the processor go to the interrupt routine after the current instruction is executed.

– **RESET : =0** Starts the reset sequence.

1.6.4 Maximum Mode Interface

• For multiprocessor environment

• 8288 Bus Controller is used for bus control

• WR¯,IO/M¯,DT/R¯,DEN¯,ALE, INTA¯ signals are not available

- Instead

– **MRDC¯** (memory read command)

– **MWRT¯** (memory write command)

– **AMWC¯** (advanced memory write command)

– **IORC¯** (I/O read command)

– **IOWC¯** (I/O write command)

– **AIOWC¯** (Advanced I/O write command)

– **INTA¯** (interrupt acknowledge*)*

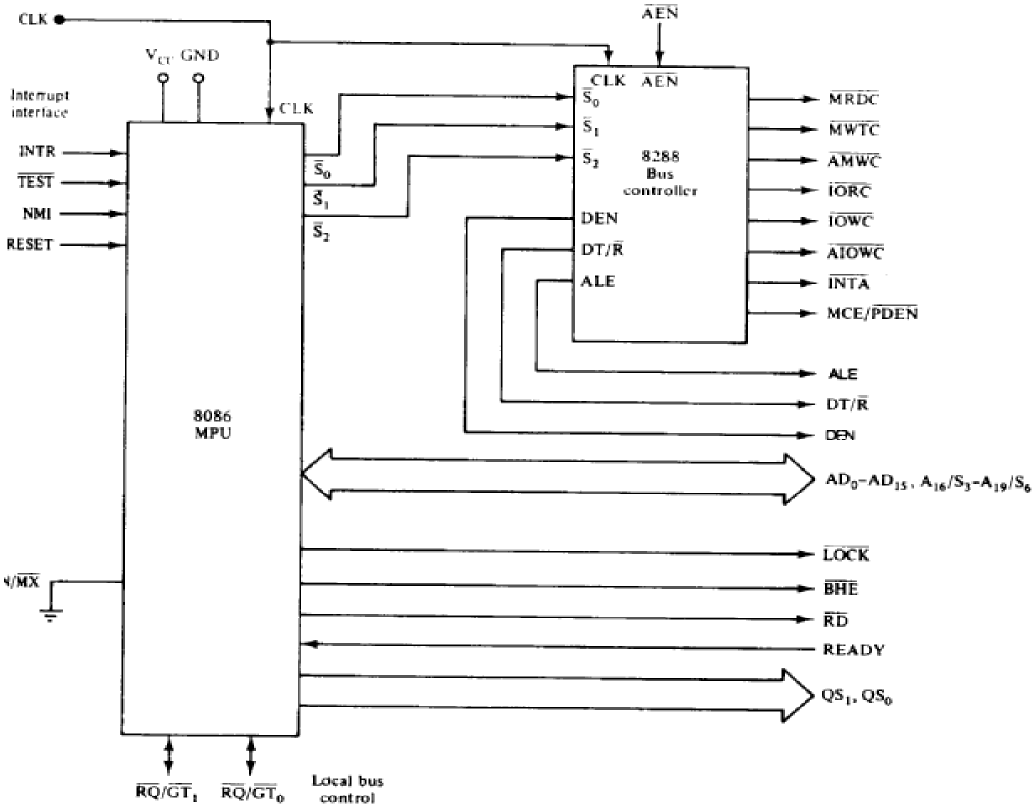


Fig 1.21 Maximum Mode Interface

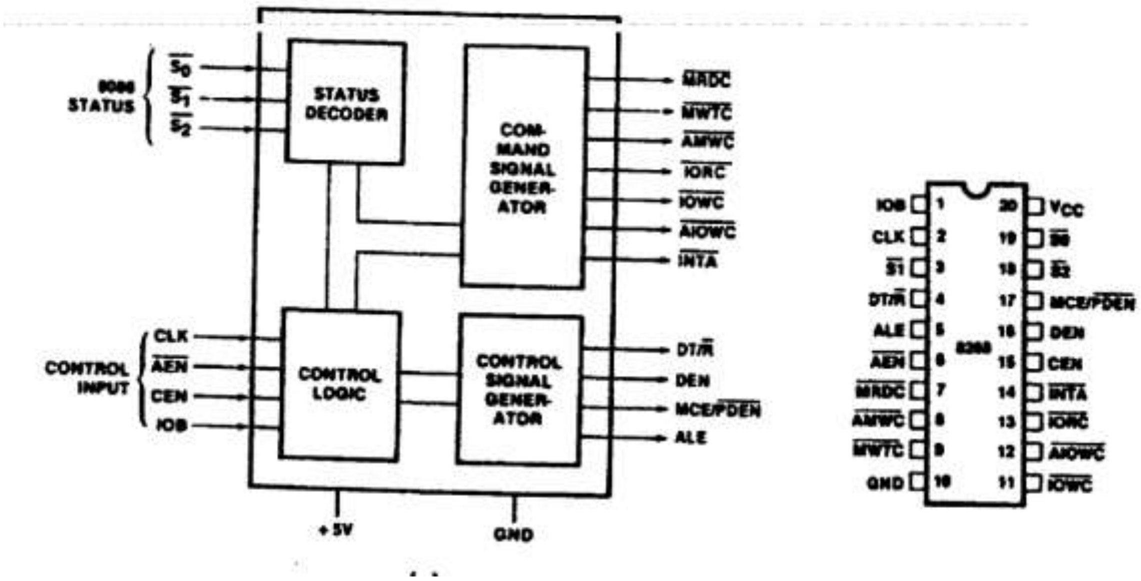


Fig 1.22Block and Pin Diagram of 8288 Bus controller

– The signals shown above are produced by 8288 depending on the state of S0, S1 and S2.

• **DEN, DT/R¯** and ALE signals are the same as minimum-mode systems

• **LOCK¯:** when =0, prevents other processors from using the bus

• QS0 and QS1 (queue status signals) : informs about the status of the queue

• **RQ¯/GT ¯0** and **RQ¯/GT ¯1** are used instead of HOLD and HLDA lines in a multiprocessor environment as request/grant lines.

1.7I/O Ports

There are two methods in which I/O devices can be connected to the Microprocessor.

(i)Memory mapped I/O

(ii)I/O mapped I/O

(i) Memory mapped I/O

In this method I/O device is treated like the memory.Here there is no IO/M signal.If the processor wants to read the data from a I/O device it will place the address of the I/O device on the address bus.Then the I/O device will get selected.The memory which is having the same address wioll also get selected.so we have to use separate address for memory and separate address for I/O device.

(ii)I/O mapped I/O

Here we hve the IO/M signal.So we can select either the memory or I/O device for read and write operation.

1.8 Data Transfer Concepts

(i)**Parallel data transfer**

(ii)Serial data transfer

(i)Parallel Data transfer

(a)Programmed I/O

(b) Interrupt I/O

(C) DMA

(a)Programmed I/O

Here the processor has to check whether the I/O device is ready or not through the Ready signal of the I/O device.If the ready signal is high then it will send the data to the I/O device.Otherwise it will continuously check theReady signal.The processor is busy in checking the Ready signal.The draw back is wastage of time.

(b) Interrupt I/O

In this method the I/O device will interrupt the Processor through the INTR signal to indicate to the processor that it is ready to accept the next data.Then the processor will send the INTA signal.Then the processor stops its normal execution and start transferring the data to the I/O device.

(c)DMA

Using DMA I/O device can directly transfer the data to the Memory using the Address and Data buses of Processor.

(ii)Serial data Transfer

Some of the external I/0 devices receive only the serial data.Normally serial communication is used in the Multi Processor environment.8051 has two pins for serial communication.

(1)SID- Serial Input data.

(2)SOD-Serial Output data.

UNIT II

PROGRAMMING OF 8085 MICROPROCESSOR

2.1 Instruction Format

*An **instruction** is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts: one is task to be performed, called the **operation code** (opcode), and the second is the data to be operated on, called the **operand.** The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address. In some instructions, the operand is implicit.*

Instruction word size

The 8085 instruction set is classified into the following three groups according to word size:

1. One-word or 1-byte instructions

2. Two-word or 2-byte instructions

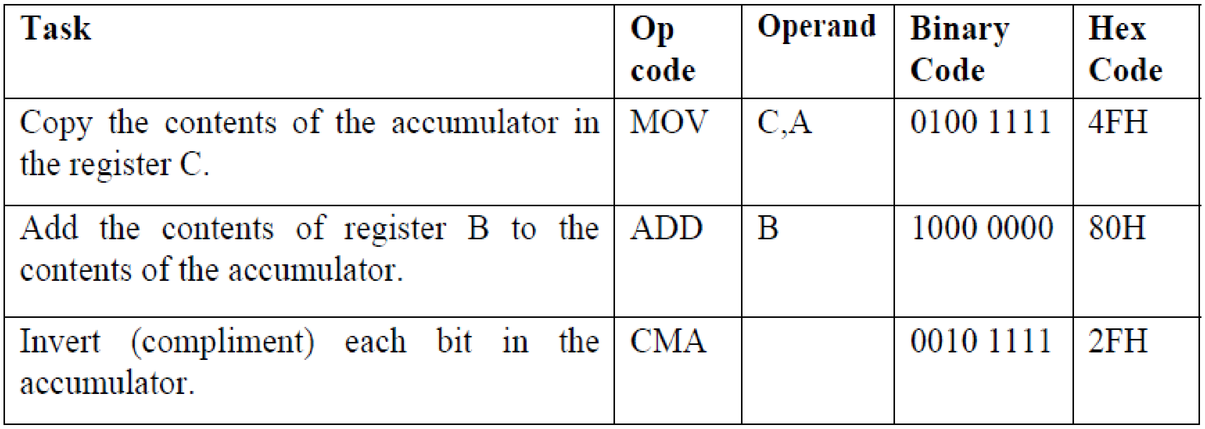
3. Three-word or 3-byte instructions

In the 8085, "byte" and "word" are synonymous because it is an 8-bit microprocessor. However, instructions are commonly referred to in terms of bytes rather than words.

2.1.1 One-Byte Instructions

A 1-byte instruction includes the opcode and operand in the same byte. Operand(s) are internal register and are coded into the instruction.

Table 2.1 Example for 1 byte Instruction



Task	Op code	Operand	Binary Code	Hex Code
Copy the contents of the accumulator in the register C.	MOV	C,A	0100 1111	4FH
Add the contents of register B to the contents of the accumulator.	ADD	B	1000 0000	80H
Invert (compliment) each bit in the accumulator.	CMA		0010 1111	2FH

These instructions are 1-byte instructions performing three different tasks. In the first instruction, both operand registers are specified. In the second instruction, the operand B is specified and the accumulator is assumed. Similarly, in the third instruction, the accumulator is assumed to be the implicit operand. These instructions are stored in 8-bit binary format in memory; each requires one memory location.

MOV rd, rs

rd <-- rs copies contents of rs into rd.

Coded as 01 ddd sss where ddd is a code for one of the 7 general registers which is the destination of the data, sss is the code of the source register.

Example: MOV A,B

Coded as 01111000 = 78H = 170 octal (octal was used extensively in instruction design of such processors).

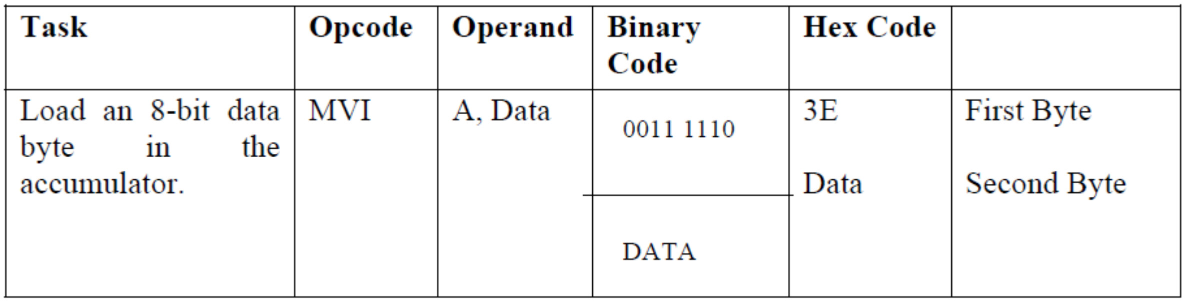
ADD r

A <-- A + r

2.1.2 Two-Byte Instructions

In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode. For example*:*

Table 2.2 Example for 2 byte Instruction



Task	Opcode	Operand	Binary Code	Hex Code	
Load an 8-bit data byte in the accumulator.	MVI	A, Data	0011 1110	3E	First Byte
			DATA	Data	Second Byte

The instruction would require two memory locations to store in memory.

MVI r,data

r <-- data

Example: MVI A,30H coded as 3EH 30H as two contiguous bytes. This is an example of immediate addressing.

ADI data

A <-- A + data

OUT port

0011 1110

DATA

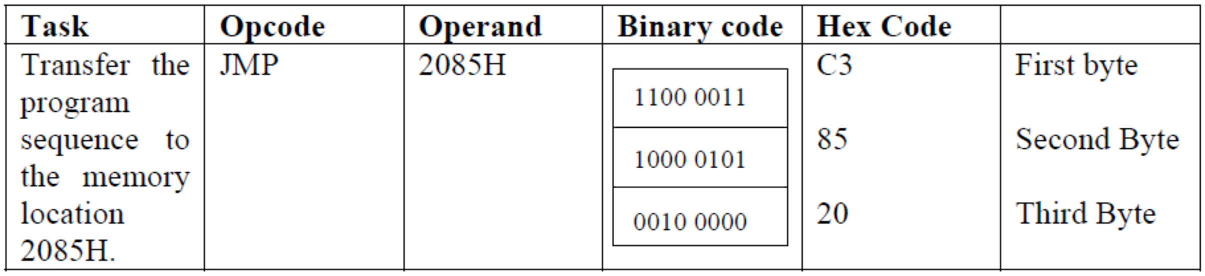
where port is an 8-bit device address. (Port) <-- A. Since the byte is not the data but points directly to where it is located this is called direct addressing.

2.1.3 Three-Byte Instructions

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address.

opcode + data byte + data byte

Table 3.3 Example for 3 byte Instruction



Task	Opcode	Operand	Binary code	Hex Code	
Transfer the program sequence to the memory location 2085H.	JMP	2085H	1100 0011	C3	First byte
			1000 0101	85	Second Byte
			0010 0000	20	Third Byte

This instruction would require three memory locations to store in memory.

Three byte instructions - opcode + data byte + data byte

LXI rp, data16

rp is one of the pairs of registers BC, DE, HL used as 16-bit registers. The two data bytes are 16-bit data in L H order of significance.

rp <-- data16

LXI H,0520H coded as 21H 20H 50H in three bytes. This is also immediate addressing.

LDA addr

A <-- (addr) Addr is a 16-bit address in L H order.

Example: LDA 2134H coded as

3AH 34H 21H. This is also an example of direct addressing.

2.2 The 8085 Addressing Modes

The instructions MOV B, A or MVI A, 82H are to copy data from a source into a destination. In these instructions the source can be a register, an input port, or an 8-bit number (00H to FFH). Similarly, a destination can be a register or an output port. The sources and destination are operands. The various formats for specifying operands are called the ADDRESSING MODES. For 8085, they are:

1. Immediate addressing.
2. Register addressing.
3. Direct addressing.
4. Indirect addressing.

(1)Immediate addressing

Data is present in the instruction. Load the immediate data to the destination provided.

Example: MVI R,data

(2)Register addressing

Data is provided through the registers.

Example: MOV Rd, Rs

(3)Direct addressing

Used to accept data from outside devices to store in the accumulator or send the data stored in the accumulator to the outside device. Accept the data from the port 00H and store them into the accumulator or Send the data from the accumulator to the port 01H.

Example: IN 00H or OUT 01H

(4)Indirect Addressing

This means that the Effective Address is calculated by the processor. And the contents of the address (and the one following) is used to form a second address. The second address is where the data is stored. Note that this requires several memory accesses; two accesses to retrieve the 16-bit address and a further access (or accesses) to retrieve the data which is to be loaded into the register.

2.3 Instruction Set Classification

An **instruction** is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions, called the **instruction set**, determines what functions the microprocessor can perform. These instructions can be classified into the following five functional categories: data transfer (copy) operations, arithmetic operations, logical operations, branching operations, and machine-control operations.

2.3.1 Data Transfer Croup

The data transfer instructions move data between registers or between memory and registers.

MOV	Move
MVI	Move Immediate
LDA	Load Accumulator Directly from Memory
STA	Store Accumulator Directly in Memory
LHLD	Load H & L Registers Directly from Memory

SHLD	Store H & L Registers Directly in Memory

An 'X' in the name of a data transfer instruction implies that it deals with a register pair (16-bits);

LXI	Load Register Pair with Immediate data
LDAX	Load Accumulator from Address in Register Pair
STAX	Store Accumulator in Address in Register Pair
XCHG	Exchange H & L with D & E
XTHL	Exchange Top of Stack with H & L

2.3.2 Arithmetic Group

The arithmetic instructions add, subtract, increment, or decrement data in registers or memory.

ADD	Add to Accumulator
ADI	Add Immediate Data to Accumulator
ADC	Add to Accumulator Using Carry Flag
ACI	Add Immediate data to Accumulator Using Carry
SUB	Subtract from Accumulator
SUI	Subtract Immediate Data from Accumulator
SBB	Subtract from Accumulator Using Borrow (Carry) Flag
SBI	Subtract Immediate from Accumulator Using Borrow (Carry) Flag
INR	Increment Specified Byte by One
DCR	Decrement Specified Byte by One
INX	Increment Register Pair by One
DCX	Decrement Register Pair by One
DAD	Double Register Add; Add Content of Register Pair to H & L Register Pair

2.3.3 Logical Group

This group performs logical (Boolean) operations on data in registers and memory and on condition flags.

The logical AND, OR, and Exclusive OR instructions enable you to set specific bits in the accumulator ON or OFF.

ANA	Logical AND with Accumulator
ANI	Logical AND with Accumulator Using Immediate Data
ORA	Logical OR with Accumulator
OR	Logical OR with Accumulator Using Immediate Data
XRA	Exclusive Logical OR with Accumulator
XRI	Exclusive OR Using Immediate Data

The Compare instructions compare the content of an 8-bit value with the contents of the accumulator;

CMP	Compare
CPI	Compare Using Immediate Data

The rotate instructions shift the contents of the accumulator one bit position to the left or right:

RLC	Rotate Accumulator Left
RRC	Rotate Accumulator Right
RAL	Rotate Left Through Carry
RAR	Rotate Right Through Carry

Complement and carry flag instructions:

CMA	Complement Accumulator
CMC	Complement Carry Flag
STC	Set Carry Flag

2.3.4 Branch Group

The branching instructions alter normal sequential program flow, either unconditionally or conditionally. The unconditional branching instructions are as follows:

JMP	Jump
CALL	Call
RET	Return

Conditional branching instructions examine the status of one of four condition flags to determine whether the specified branch is to be executed. The conditions that may be specified are as follows:

NZ	Not Zero (Z =	0)
Z	Zero (Z = 1)	
NC	No Carry (C =	0)
C	Carry (C = 1)	
PO	Parity Odd (P	= 0)
PE	Parity Even (P	= 1)
P	Plus (S = 0)	
M	Minus (S = 1)	

Thus, the conditional branching instructions are specified as follows:

Jumps	Calls	Returns	
C	CC	RC	(Carry)
INC	CNC	RNC	(No Carry)
JZ	CZ	RZ	(Zero)
JNZ	CNZ	RNZ	(Not Zero)
JP	CP	RP	(Plus)
JM	CM	RM	(Minus)
JPE	CPE	RPE	(Parity Even)
JP0	CPO	RPO	(Parity Odd)

Two other instructions can affect a branch by replacing the contents or the program counter:

PCHL	Move H & L to Program Counter
RST	Special Restart Instruction Used with Interrupts

2.3.5 Stack Instructions

The following instructions affect the Stack and/or Stack Pointer

PUSH	Push Two bytes of Data onto the Stack

POP	Pop Two Bytes of Data off the Stack
XTHL	Exchange Top of Stack with H & L
SPHL	Move content of H & L to Stack Pointer

2.3.6 I/0 instructions

IN	Initiate Input Operation
OUT	Initiate Output Operation

2.3.7 Machine Control instructions

EI	Enable Interrupt System
DI	Disable Interrupt System
HLT	Halt
NOP	No Operation

2.4 Sample Program

(1)Write an assembly program to add two numbers Program

```
MVI D, 8BH
MVI C, 6FH
MOV A, C
1100 0011
1000 0101
0010 0000
ADD D
OUT PORT1
HLT
```

(2)Write an assembly program to multiply a number by 8 Program

```
MVI A, 30H
RRC
RRC
RRC
OUT PORT1
HLT
```

(3)Write an assembly program to find greatest between two numbers Program

```
MVI B, 30H
MVI C, 40H
MOV A, B
CMP C
JZ EQU
JC GRT
OUT PORT1
```

```
HLT
EQU: MVI A, 01H
OUT PORT1
HLT
GRT: MOV A, C
OUT PORT1
HLT
```

2.5 Programming using Loop structure with Counting and Indexing

(i) 16 bit Multiplication

ADDRESS	LABEL	MNEMONICS	OPCODE
	START	LHLD 4200	
		SPHL	
		LHLD 4202	
		XCHG	
		LXI H,0000	
	L1	LXI B,0000	
		DAD SP	
		JNC L2	
		INX B	
	L2	DCX D	
		MOV A,E	
		ORA D	
		JNZ L1	
		SHLD 4204	
		MOV L,C	

		MOV H,B	
		SHLD 4206	
		HLT	

(ii)Finding the maximum number in the given array

ADDRESS	LABEL	MNEMONICS	OPCODE
	START	LDA 4500	
		MOV C, A	
		LXI H, 4501	
	L2	MOV A, M	
	L3	DCR C	
		INX H	
		JZ L1	
		CMP M	
		JC L2	
		JMP L3	
	L1	STA 4520	
		HLT	

(iii) To sort the array of data in ascending order

ADDRESS	LABEL	MNEMONICS

	START	
	L3	MVI B, 00
		LXI H, 4200
		MOV C, M
		DCR C
		INX H
	L2	MOV A, M
		INX H
		CMP M
		JC L1
		MOV D, M
		MOV M, A
		DCX H
		MOV M, D
		INX H
		MVI B, 01
	L1	DCR C
		JNZ L2
		DCR B
		JZ L3
		HLT

2.6 Programming using subroutine Instructions

Generation of Square waveform using DAC

ADDRESS	LABEL	MNEMONICS
	START	MVI A,00H
		OUT C8
		CALL DELAY
		MVI A,FF
		OUT C8
		CALL DELAY
		JMP START
		MVI B,05H
		MVI C,FF
	DELAY	DCR C
	L2	JNZ L1
	L1	DCR B
		JNL L2
		RET

2.7 Programming using Look up table

ADDRESS	LABEL	MNEMONICS	OPCODE
	START	MVI B,08	
		MVI A,00(DISPLAY MODE SETUP)	
		OUT C2	
		MVI A,CC(CLEAR DISPLAY)	
		OUT C2	
		MVI A,90(WRITE DISPLAY RAM)	
		OUT C2	
		MVI A, FF(CLEAR DISPLAY RAM)	
		OUT C0	
		DCR B	
		JNZ L1	
		IN C2	
	L2	ANI 07	
		JZ L2	
		MVI A, 40(SET TO READ FIFO RAM)	
		OUT C2	
		IN C0	
		ANI 0F	

		MOV L, A MVI H, 42 MOV A, M OUT C0 JMP L2	

LOOKUP TABLE

4200	0C	9F	4A	0B
4204	99	29	28	8F
4208	08	09	88	38
420C	6C	1A	68	E8

UNIT III

PERIPHERAL INTERFACING

3.1 Programmable peripheral interface(8255)

3.1.1 Architecture of 8255

The parallel input-output port chip 8255 is also called as programmable peripheral input-output port. The Intel’s 8255 is designed for use with Intel’s 8-bit, 16-bit and higher capability microprocessors. It has 24 input/output lines which may be individually programmed in two groups of twelve lines each, or three groups of eight lines. The two groups of I/O pins are named as Group A and Group B. Each of these two groups contains a subgroup of eight I/O lines called as 8-bit port and another subgroup of four lines or a 4-bit port. Thus Group A contains an 8-bit port A along with a 4-bit port. C upper.The port A lines are identified by symbols PA0-PA7 while the port C lines are identified as PC4-PC7. Similarly, GroupB contains an 8-bit port B, containing lines PB0-PB7 and a 4-bit port C with lower bits PC0- PC3. The port C upper and port C lower can be used in combination as an 8-bit port C. Both the port C are assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit ports from 8255. All of these ports can function independently either as input or as output ports. This can be achieved by programming the bits of an internal register of 8255 called as control word register (CWR). This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor. The control words or status information is also transferred through the buffer.

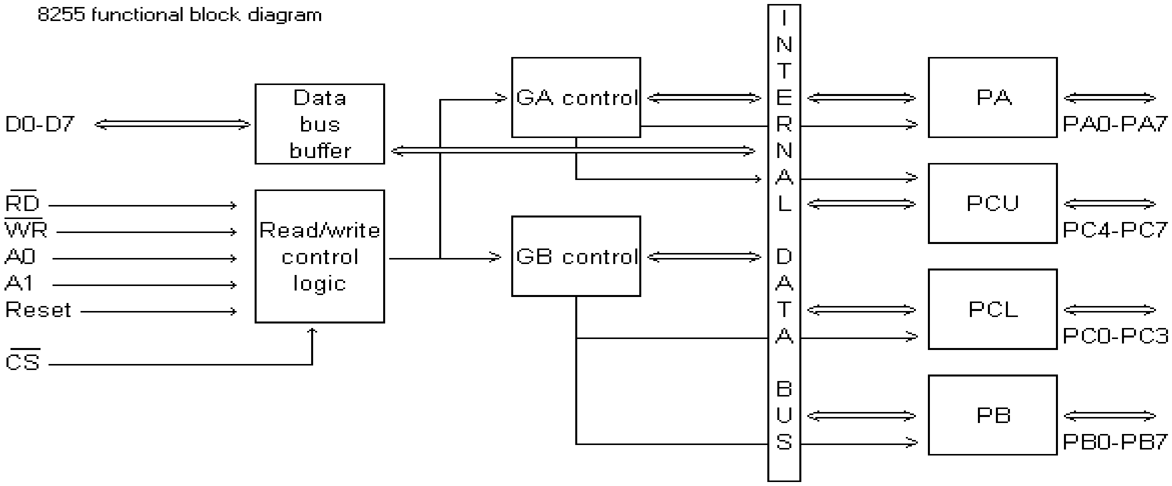


Fig 3.1 8255 Architecture

3.1.2 Pin Diagram of 8255

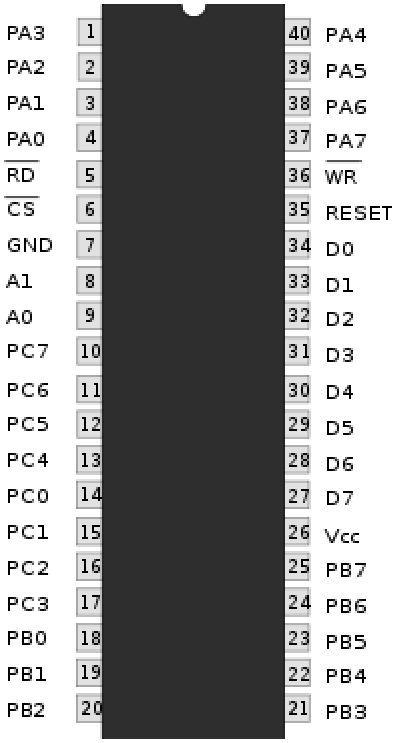


Fig 3.2 Pin Diagram of 8255

The signal description of 8255 are briefly presented as follows :

• **PA7-PA0**: These are eight port A lines that acts as either latched output or buffered input lines depending upon the control word loaded into the control word register.

• **PC7-PC4** : Upper nibble of port C lines. They may act as either output latches or input buffers lines.This port also can be used for generation of handshake lines in mode 1 or mode 2.

• **PC3-PC0** : These are the lower port C lines, other details are the same as PC7-PC4 lines.

• **PB0-PB7** : These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.

• **RD** : This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.

• **WR** : This is an input line driven by the microprocessor. A low on this line indicates write operation.

CS : This is a chip select line. If this line goes low, it enables the 8255 to respond to RD and WR signals, otherwise RD and WR signal are neglected.

• **A1-A0** : These are the address input lines and are driven by the microprocessor. These lines A1-A0 with RD, WR and CS from the following operations for 8255. These address lines are used for addressing any one of the four registers,i.e. three ports and a control word register as given in table below.

• In case of 8086 systems, if the 8255 is to be interfaced with lower order data bus, the A0 and A1 pins of 8255 are connected with A1 and A2 respectively.

D0-D7 : These are the data bus lines those carry data or control word to/from the microprocessor.

• **RESET** : A logic high on this line clears the control word register of 8255. All ports are set as input ports by default after reset.

3.1.3 Operational Modes of 8255

There are two main operational modes of 8255:

1. Input/output mode 2. Bit set/reset mode

3.1.3.1 Input/Output Mode

There are three types of the input/output mode. They are as follows:

- **Mode 0**

In this mode, the ports can be used for simple input/output operations without handshaking. If both port A and B are initialized in mode 0, the two halves of port C can be either used together as an additional 8-bit port, or they can be used as individual 4-bit ports. Since the two halves of port C are independent, they may be used such that one-half is initialized as an input port while the other half is initialized as an output port. The input output features in mode 0 are as follows: 1. O/p are latched. 2. I/p are buffered not latched. 3. Port do not have handshake or interrupt capability.

- **Mode 1**

When we wish to use port A or port B for handshake (strobed) input or output operation, we initialise that port in mode 1 (port A and port B can be initilalised to operate in different modes,ie, for eg, port A can operate in mode 0 and port B in mode 1). Some of the pins of port C function as handshake lines.

For port B in this mode (irrespective of whether is acting as an input port or output port), PC0, PC1 and PC2 pins function as handshake lines.

If port A is initialised as mode 1 input port, then, PC3, PC4 and PC5 function as handshake signals. Pins PC6 and PC7 are available for use as input/output lines.

The mode 1 which supports handshaking has following features: 1. Two ports i.e. port A and B can be use as 8-bit i/o port. 2. Each port uses three lines of port c as handshake signal and remaining two signals can be function as i/o port. 3. interrupt logic is supported. 4. Input and Output data are latched.

- **Mode 2**

Only group A can be initialised in this mode. Port A can be used for bidirectional handshake data transfer. This means that data can be input or output on the same eight lines (PA0 - PA7). Pins PC3 - PC7 are used as handshake lines for port A. The remaining pins of port C (PC0 - PC2) can be used as input/output lines if group B is initialised in mode 0. In this mode, the 8255 may be used to extend the system bus to a slave microprocessor or to transfer data bytes to and from a floppy disk controller.

3.1.3.2 Bit Set/Reset (BSR) mode

In this mode only port b can be used (as an output port). Each line of port C (PC0 - PC7) can be set/reset by suitably loading the command word register.no effect occurs in input-output mode. The individual bits of port c can be set or reset by sending the signal OUT instruction to the control register.

3.1.4 Control Word Format

3.1.4.1 Input/output mode format

D7	D6	D5	D4	D3	D2	D1	D0
1 (1=I/O)	GA mode select		PA	PCU	GB mode select	PB	PCL

Fig 3.3 Control Word format for Input/Output Mode

Control Word format in input/output mode

- The figure shows the control word format in the input/output mode. This mode is selected by making **D7 = '1'** .

- **D0, D1, D3, D4** are for lower port C, port B, upper port C and port A respectively. When D0 or D1 or D3 or D4 are "SET", the corresponding ports act as input ports. For eg, if D0 = D4 = '1', then lower port C and port A act as input ports. If these bits are "RESET", then the corresponding ports act as output ports. For eg, if D1 = D3 = '0', then port B and upper port C act as output ports.

- **D2** is used for mode selection for group B (Port B and Lower Port C). When D2 = '0', mode 0 is selected and when D2 = '1', mode 1 is selected.

- **D5, D6** are used for mode selection for group A (Upper Port C and Port A). The format is as follows:

D6	D5	mode
0	0	0
0	1	1
1	x	2

3.1.4.2 BSR mode format

Control Word format in BSR mode

D7	D6	D5	D4	D3	D2	D1	D0
0 (0=BSR)	X	X	X	B2	B1	B0	S/R (1=S,0=R)

Bit select: (Taking Don't care's as 0)

B2	B1	B0	PC bit	Control word (Set)	Control word (reset)
0	0	0	0	0000 0001 = 01h	0000 0000 = 00h
0	0	1	1	0000 0011 = 03h	0000 0010 = 02h
0	1	0	2	0000 0101 = 05h	0000 0100 = 04h
0	1	1	3	0000 0111 = 07h	0000 0110 = 06h
1	0	0	4	0000 1001 = 09h	0000 1000 = 08h
1	0	1	5	0000 1011 = 0Bh	0000 1010 = 0Ah
1	1	0	6	0000 1101 = 0Dh	0000 1100 = 0Ch
1	1	1	7	0000 1111 = 0Fh	0000 1110 = 0Eh

Fig 3.4 Control Word format in BSR mode

- The figure shows the control word format in BSR mode. This mode is selected by making **D7='0'**.

- **D0** is used for bit set/reset. When D0= '1', the port C bit selected (selection of a port C bit is shown in the next point) is **SET**, when D0 = '0', the port C bit is **RESET**.

- **D1, D2, D3** are used to select a particular port C bit whose value may be altered using D0 bit as mentioned above. The selection of the port C bits are done as follows:

D3	D2	D1	bit/pin of port C selected
0	0	0	PC0
0	0	1	PC1
0	1	0	PC2
0	1	1	PC3
1	0	0	PC4
1	0	1	PC5
1	1	0	PC6
1	1	1	PC7

- **D4, D5, D6** are not used.

3.2Programmable Interrupt Controller(8259)

3.2.1 Features

- 8 levels of interrupts.
- Can be cascaded in master-slave configuration to handle 64 levels of interrupts.
- Internal priority resolver.
- Fixed priority mode and rotating priority mode.
- Individually maskable interrupts.
- Modes and masks can be changed dynamically.
- Accepts IRQ, determines priority, checks whether incoming priority > current level being serviced, issues interrupt signal.
- In 8085 mode, provides 3 byte CALL instruction. In 8086 mode, provides 8 bit vector number.
- Polled and vectored mode.
- Starting address of ISR or vector number is programmable.
- No clock required.

3.2.2 *Pinout*

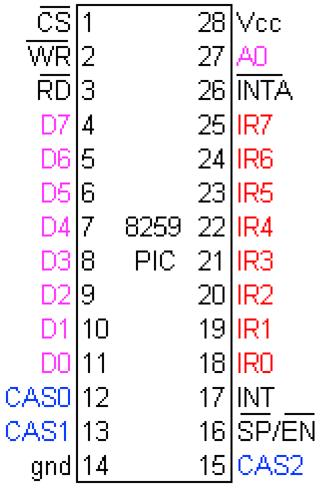


Fig 3.5 Pin Diagram of 8259

Table 3.1 Pin Description of 8259

D0-D7	Bi-directional, tristated, buffered data lines. Connected to data bus directly or through buffers
RD-bar	Active low read control
WR-bar	Active low write control
A0	Address input line, used to select control register
CS-bar	Active low chip select
CAS0-2	Bi-directional, 3 bit cascade lines. In master mode, PIC places slave ID no. on these lines. In slave mode, the PIC reads slave ID no. from master on these lines. It may be regarded as slave-select.
SP-bar / EN-bar	Slave program / enable. In non-buffered mode, it is SP-bar input, used to distinguish master/slave PIC. In buffered mode, it is output line used to enable buffers
INT	Interrupt line, connected to INTR of microprocessor
INTA-bar	Interrupt ack, received active low from microprocessor
IR0-7	Asynchronous IRQ input lines, generated by peripherals.

3.2.3 Block diagram

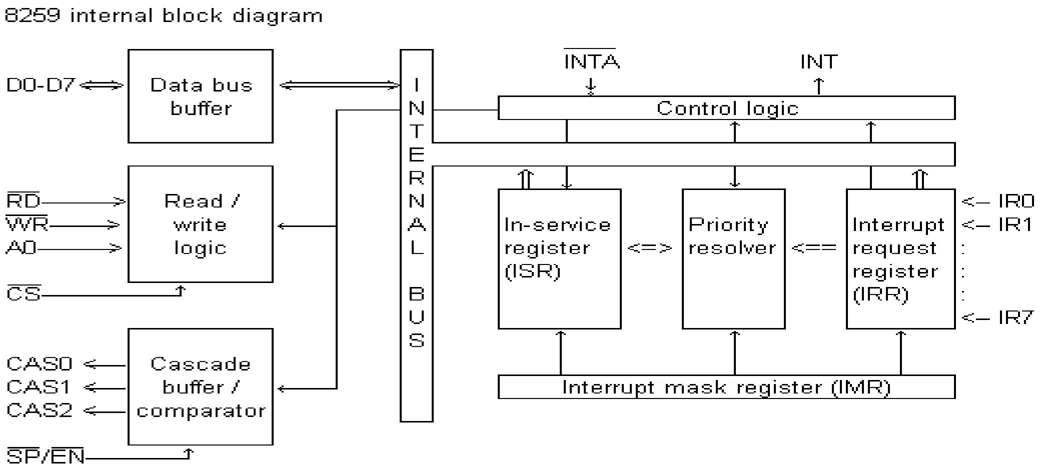

8259 internal block diagram
D0-D7
Data bus buffer
RD
WR
A0
Read / write logic
CS
CAS0
CAS1
CAS2
Cascade buffer / comparator
SP/EN
INTERNAL BUS
INTA
INT
Control logic
In-service register (ISR)
Priority resolver
Interrupt request register (IRR)
<– IR0
<– IR1
<– IR7
Interrupt mask register (IMR)

Fig 3.6 Block Diagram of 8259

ICW1 (Initialisation Command Word One)

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	A7	A6	A5	1	LTIM	ADI	SNGL	IC4

D0: IC4: 0=no ICW4, 1=ICW4 required
D1: SNGL: 1=Single PIC, 0=Cascaded PIC
D2: ADI: Address interval. Used only in 8085, not 8086. 1=ISR's are 4 bytes apart (0200, 0204, etc) 0=ISR's are 8 byte apart (0200, 0208, etc)
D3: LTIM: level triggered interrupt mode: 1=All IR lines level triggered. 0=edge triggered
D4-D7: A5-A7: 8085 only. ISR address lower byte segment. The lower byte is

A7	A6	A5	A4	A3	A2	A1	A0

of which A7, A6, A5 are provided by D7-D5 of ICW1 (if ADI=1), or A7, A6 are provided if ADI=0. A4-A0 (or A5-A0) are set by 8259 itself:

ADI=1 (spacing 4 bytes)

IRQ	A7	A6	A5	A4	A3	A2	A1	A0
IR0	A7	A6	A5	0	0	0	0	0
IR1	A7	A6	A5	0	0	1	0	0
IR2	A7	A6	A5	0	1	0	0	0
IR3	A7	A6	A5	0	1	1	0	0
IR4	A7	A6	A5	1	0	0	0	0
IR5	A7	A6	A5	1	0	1	0	0
IR6	A7	A6	A5	1	1	1	0	0
IR7	A7	A6	A5	1	1	1	0	0

ADI=0 (spacing 8 bytes)

IRQ	A7	A6	A5	A4	A3	A2	A1	A0
IR0	A7	A6	0	0	0	0	0	0
IR1	A7	A6	0	0	1	0	0	0
IR2	A7	A6	0	1	0	0	0	0
IR3	A7	A6	0	1	1	0	0	0
IR4	A7	A6	1	0	0	0	0	0
IR5	A7	A6	1	0	1	0	0	0
IR6	A7	A6	1	1	0	0	0	0
IR7	A7	A6	1	1	1	0	0	0

- **ICW2 (Initialisation Command Word Two)**

Higher byte of ISR address (8085), or 8 bit vector address (8086).

A0
1

D7	D6	D5	D4	D3	D2	D1	D0
A15	A14	A13	A12	A11	A10	A9	A8

- **ICW3 (Initialisation Command Word Three)**

A0
1

	D7	D6	D5	D4	D3	D2	D1	D0
Master	S7	S6	S5	S4	S3	S2	S1	S0
Slave	0	0	0	0	0	ID3	ID2	ID1

- Master mode: 1 indicates slave is present on that interrupt, 0 indicates direct interrupt
- Slave mode: ID3-ID2-ID1 is the slave ID number. Slave 4 on IR4 has ICW3=04h (0000 0100)

- **ICW4 (Initialisation Command Word Four)**

A0
1

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	SFNM	BUF	M/S	AEOI	Mode

- SFNM: 1=Special Fully Nested Mode, 0=FNM
- M/S: 1=Master, 0=Slave
- AEOI: 1=Auto End of Interrupt, 0=Normal
- Mode: 0=8085, 1=8086

- **OCW1 (Operational Command Word One)**

A0
1

D7	D6	D5	D4	D3	D2	D1	D0
M7	M6	M5	M4	M3	M2	M1	M0

IRn is masked by setting Mn to 1; mask cleared by setting Mn to 0 (n=0..7)

- **OCW2 (Operational Command Word Two)**

A0

D7	D6	D5	D4	D3	D2	D1	D0

1	R	SL	EOI	0	0	L3	L2	L1

	R	SL	EOI	Action
EOI	0	0	1	Non specific EOI (L3L2L1=000)
	0	1	1	Specific EOI command (Interrupt to clear given by L3L2L1)
Auto rotation of priorities (L3L2L1=000)	1	0	1	Rotate priorities on non-specific EOI
	1	0	0	Rotate priorities in auto EOI mode set
	0	0	0	Rotate priorities in auto EOI mode clear
Specific rotation of priorities (Lowest priority ISR=L3L2L1)	1	1	1	Rotate priority on specific EOI command (resets current ISR bit)
	1	1	0	Set priority (does not reset current ISR bit)
	0	1	0	No operation

- **OCW3 (Operational Command Word Three)**

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	D7	ESMM	SMM	0	1	MODE	RIR	RIS

ESMM	SMM	Effect
0	X	No effect
1	0	Reset special mask
1	1	Set special mask

3.3 8251 UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER (USART)

The 8251 is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication. As a peripheral device of a microcomputer system, the 8251receives parallel data from the CPU and transmits serial data after conversion. This device also receives serial data from the outside and transmits parallel data to the CPU after conversion.

3.3.1 Block Diagram of 8251

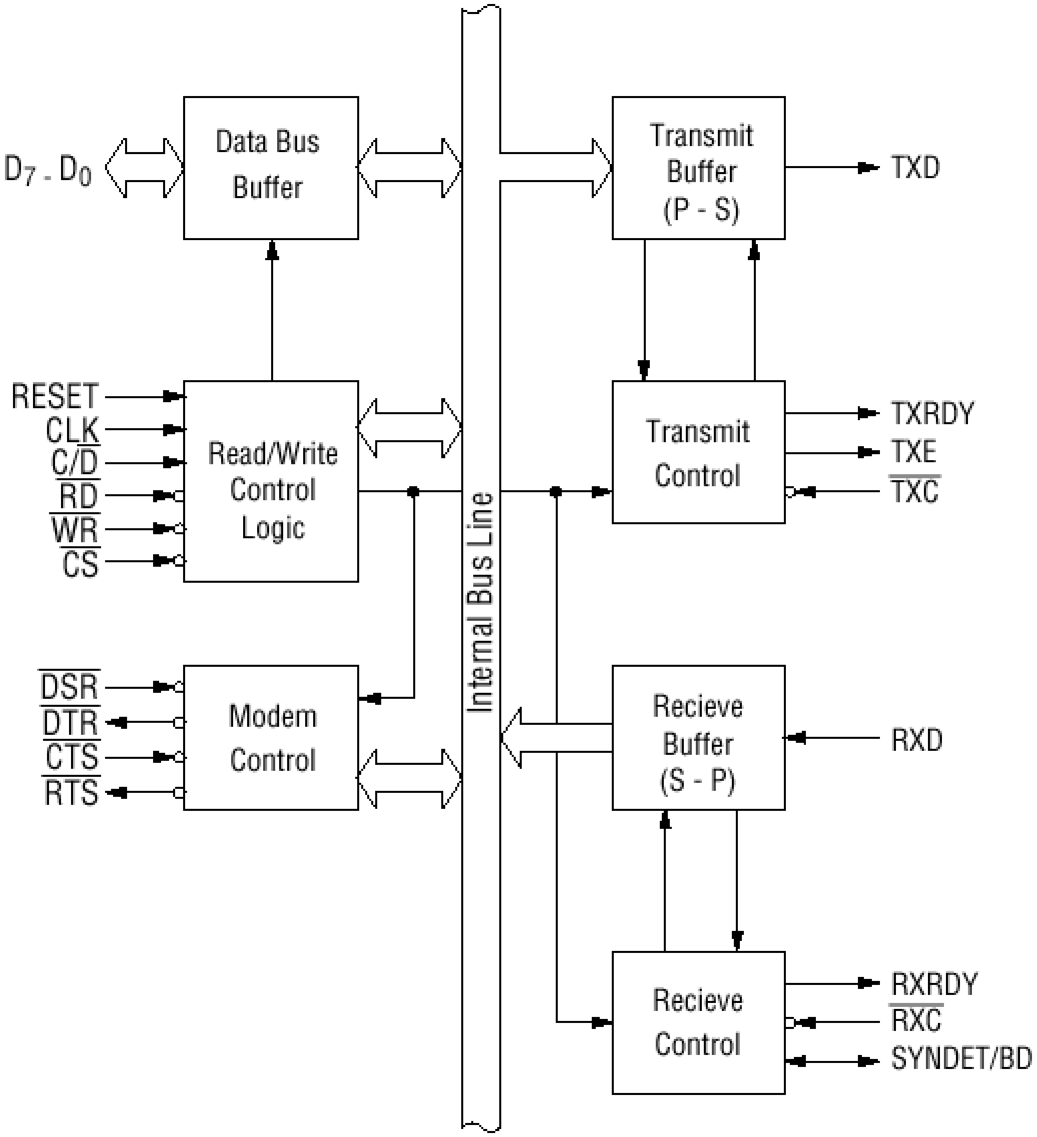


Fig 3.7 Block diagram of the 8251 USART (Universal Synchronous Asynchronous Receiver Transmitter)

3.3.2 Control Words

There are two types of control word.

1. Mode instruction (setting of function)

2. Command (setting of operation)

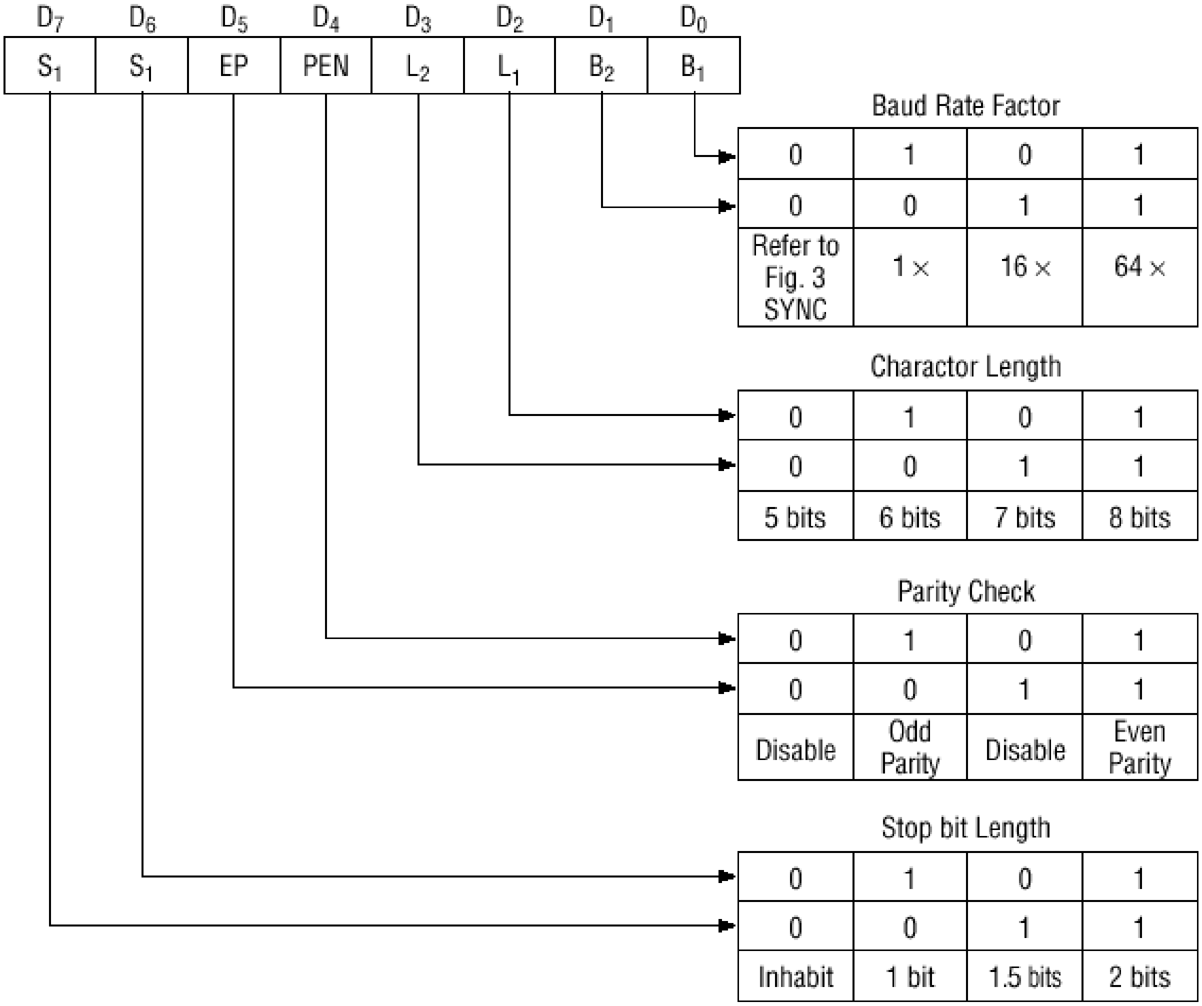
1) Mode Instruction

Mode instruction is used for setting the function of the 8251. Mode instruction will be in "wait for write" at either internal reset or external reset. That is, the writing of a control word after resetting will be recognized as a "mode instruction."

Items set by mode instruction are as follows:

- Synchronous/asynchronous mode
- Stop bit length (asynchronous mode)
- Character length
- Parity bit
- Baud rate factor (asynchronous mode)
- Internal/external synchronization (synchronous mode)
- Number of synchronous characters (Synchronous mode)

The bit configuration of mode instruction is shown in Figures 2 and 3. In the case of synchronous mode, it is necessary to write one-or two byte sync characters. If sync characters were written, a function will be set because the writing of sync characters constitutes part of mode instruction.



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S_1	S_1	EP	PEN	L_2	L_1	B_2	B_1

Baud Rate Factor

0	1	0	1
0	0	1	1
Refer to Fig. 3 SYNC	1 ×	16 ×	64 ×

Charactor Length

0	1	0	1
0	0	1	1
5 bits	6 bits	7 bits	8 bits

Parity Check

0	1	0	1
0	0	1	1
Disable	Odd Parity	Disable	Even Parity

Stop bit Length

0	1	0	1
0	0	1	1
Inhabit	1 bit	1.5 bits	2 bits

Fig 3.8 Bit Configuration of Mode Instruction(Asynchronous)

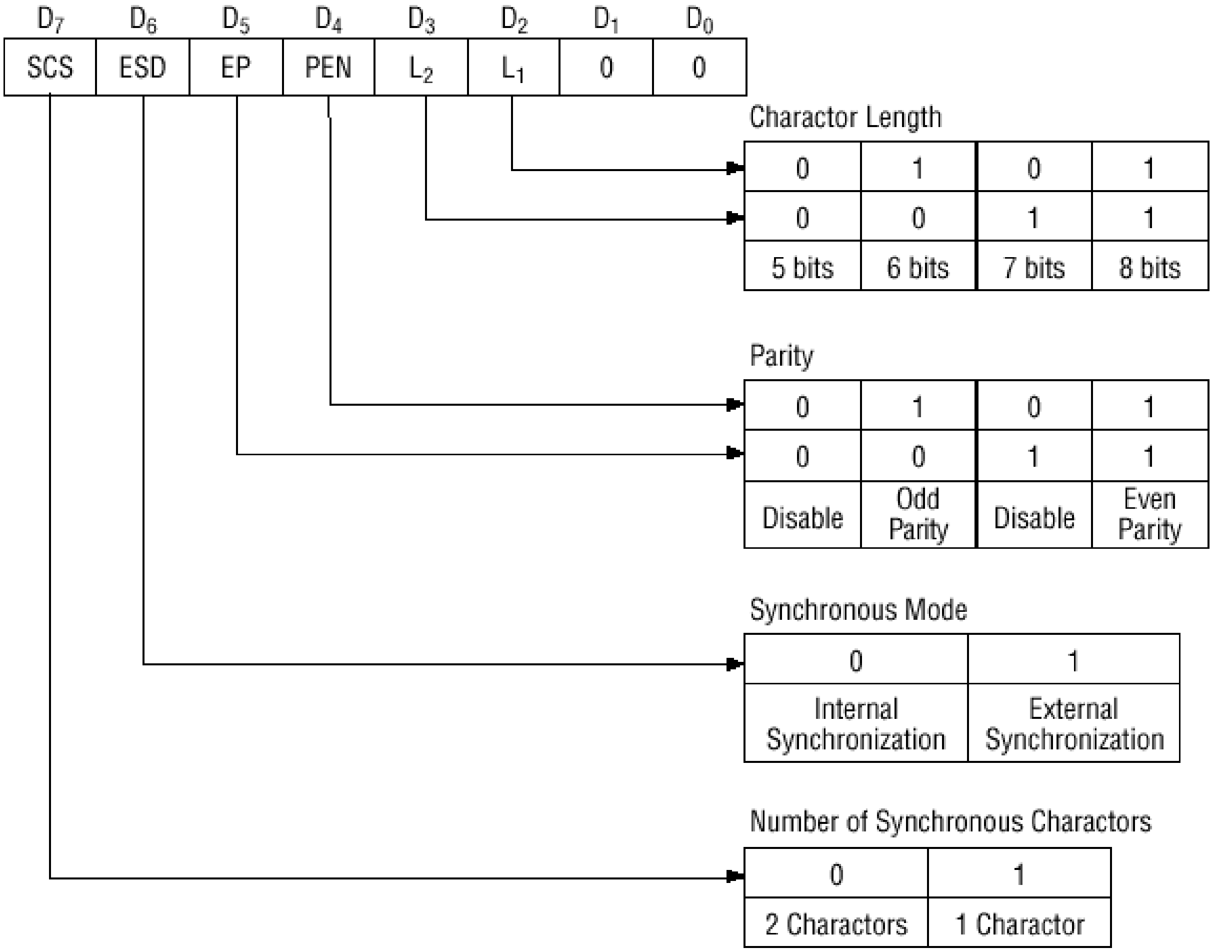


Fig 3.9 Bit Configuration of Mode Instruction(synchronous)

2) Command

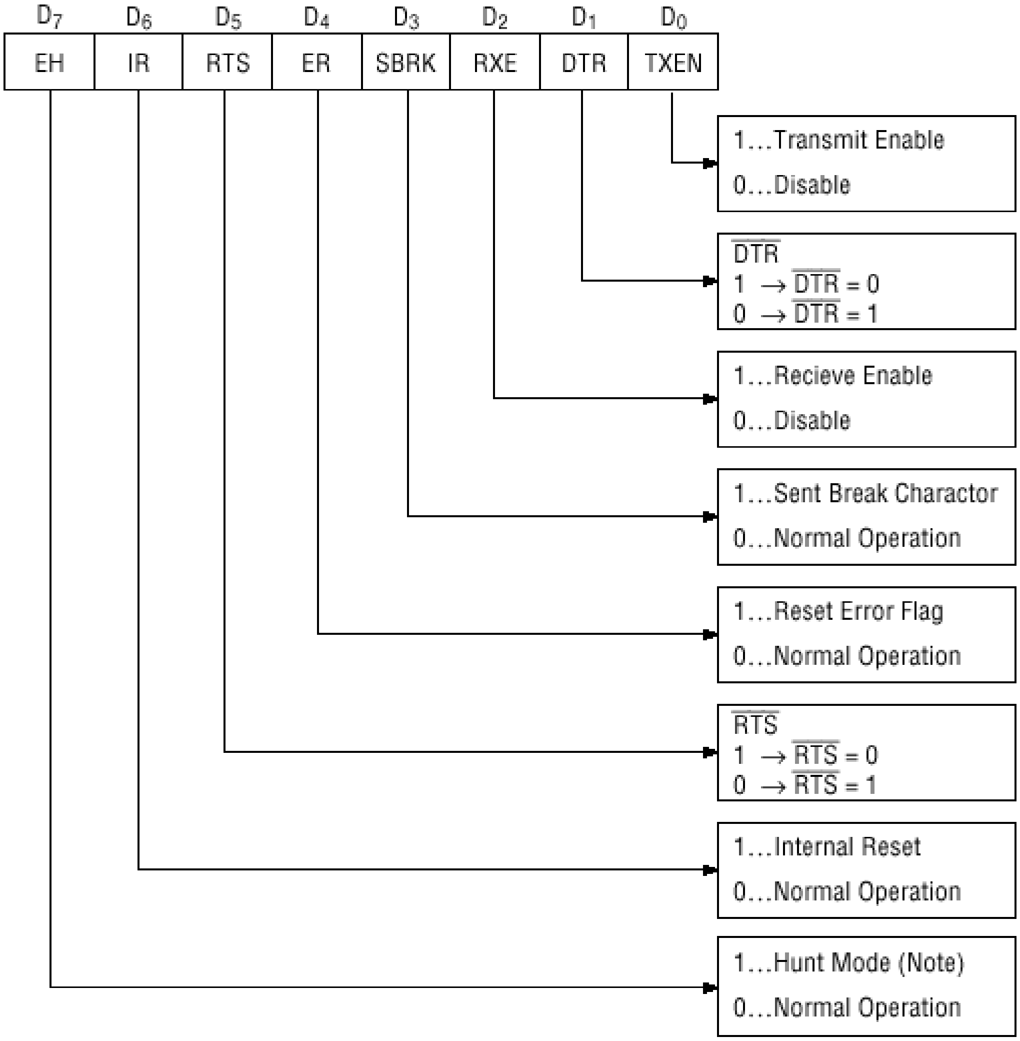
Command is used for setting the operation of the 8251. It is possible to write a command whenever necessary after writing a mode instruction and sync characters.

Items to be set by command are as follows:

- Transmit Enable/Disable

- Receive Enable/Disable

- DTR, RTS Output of data.
- Resetting of error flag.
- Sending to break characters
- Internal resetting
- Hunt mode (synchronous mode)



Note: Seach mode for synchronous charactors in synchronous mode.

Fig 3.10 Bit Configuration of Command

3.3.3 Status Word

It is possible to see the internal status of the 8251 by reading a status word.

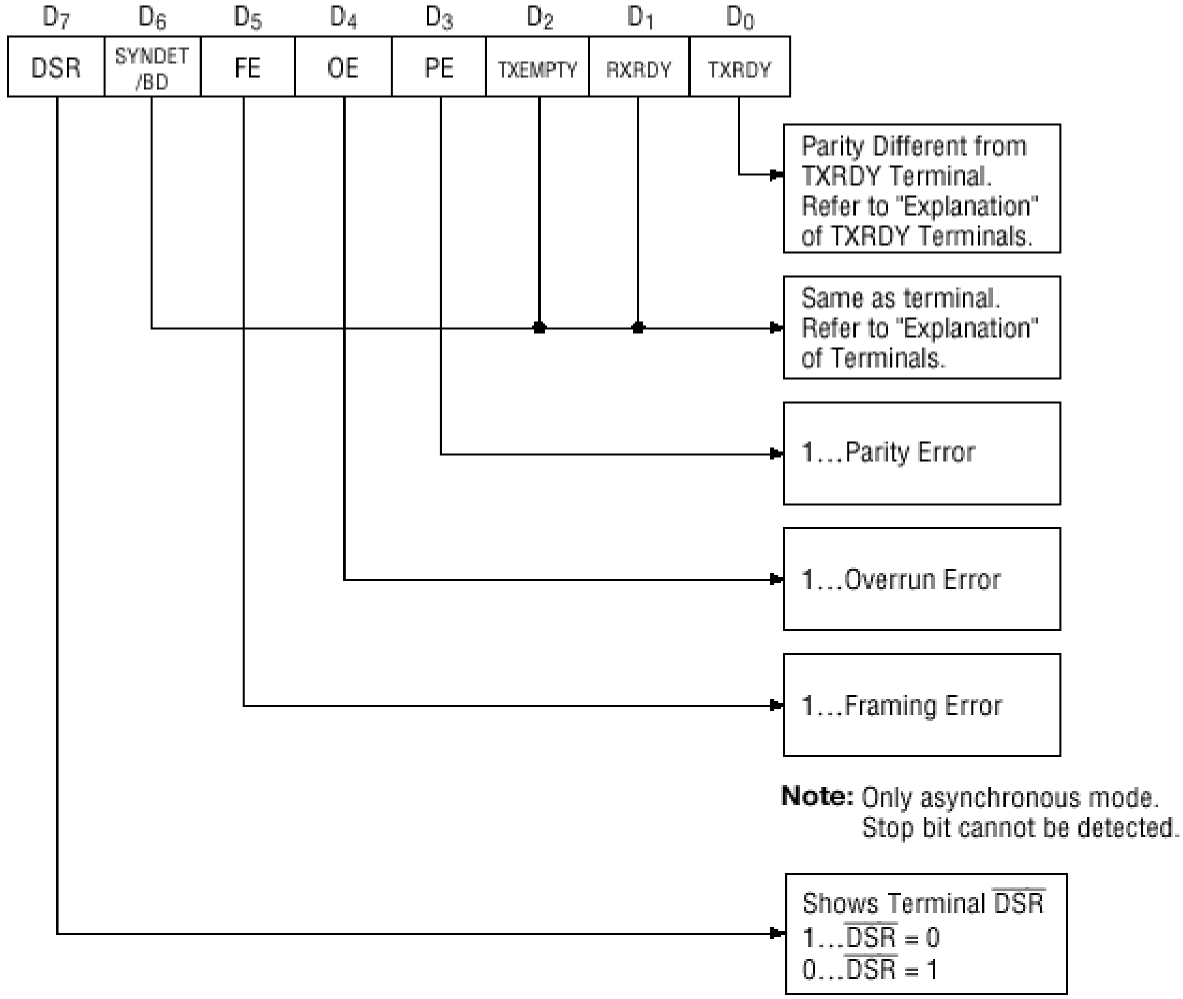

D7
D6
D5
D4
D3
D2
D1
D0
DSR
SYNDET /BD
FE
OE
PE
TXEMPTY
RXRDY
TXRDY
Parity Different from TXRDY Terminal. Refer to "Explanation" of TXRDY Terminals.
Same as terminal. Refer to "Explanation" of Terminals.
1…Parity Error
1…Overrun Error
1…Framing Error
Note: Only asynchronous mode. Stop bit cannot be detected.
Shows Terminal $\overline{DSR}$
1…$\overline{DSR}$ = 0
0…$\overline{DSR}$ = 1

Fig 3.11 Bit Configuration of Status Word

3.4 Programmable Keyboard/Display Interface - 8279

A programmable keyboard and display interfacing chip.Scans and encodes up to a 64-key keyboard.Controls up to a 16-digit numerical display.Keyboard section has a built-in FIFO 8 character buffer.The display is controlled from an internal 16x8 RAM tha stores the coded display information.

3.4.1 Pinout Definition 8279

.

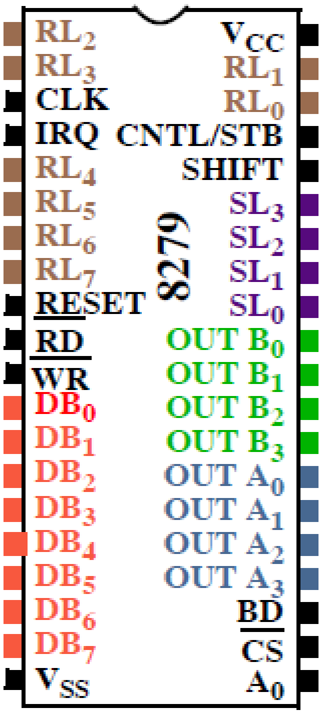


Fig 3.12 Pin Diagram of 8279

- **A0**: Selects data (0) or control/status (1) for reads and writes between micro and 8279.
- **BD:** Output that blanks the displays.
- **CLK:** Used internally for timing. Max is 3 MHz.
- **CN/ST:** Control/strobe, connected to the control key on the keyboard
- **CS**: Chip select that enables programming, reading the keyboard, etc.
- **DB7-DB0:** Consists of bidirectional pins that connect to data bus on micro.
- **IRQ:** Interrupt request, becomes 1 when a key is pressed, data is available.
- **OUT A3-A0/B3-B0**: Outputs that sends data to the most significant/least significant nibble of display.
- **RD(WR):** Connects to micro's IORC or RD signal, reads data/status registers.
- **RESET:** Connects to system RESET.
- **RL7-RL0:** Return lines are inputs used to sense key depression in the keyboard matrix.
- **Shift:** Shift connects to Shift key on keyboard.

- **SL3-SL0:** Scan line outputs scan both the keyboard and displays.

3.4.2 Block Diagram of 8279

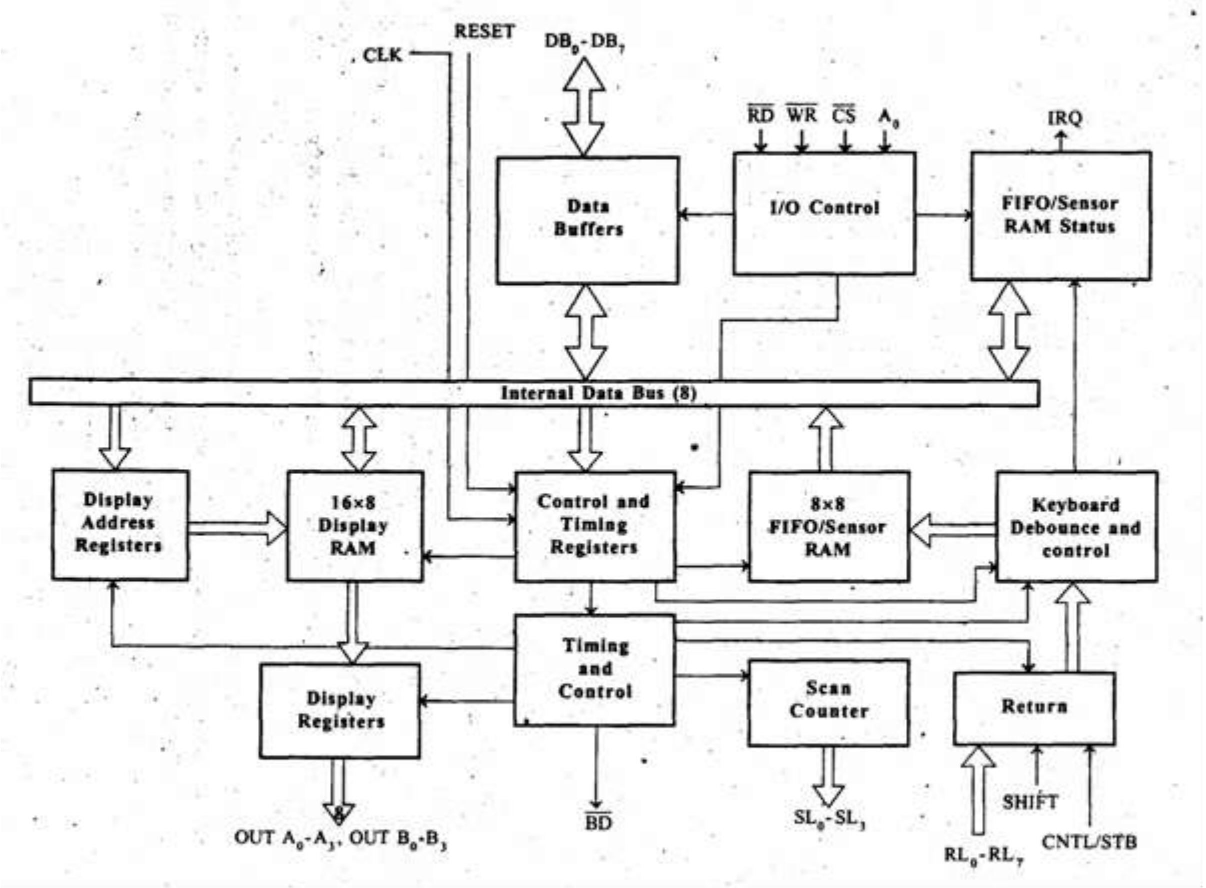


Fig 3.13 Block Diagram of 8279

Display section:

- **The display section has eight output lines divided into two groups A0-A3 and B0-B3.**
- **The output lines can be used either as a single group of eight lines or as two groups of four lines, in conjunction with the scan lines for a multiplexed display.**
- **The output lines are connected to the anodes through driver transistor in case of common cathode 7-segment LEDs.**
- **The cathodes are connected to scan lines through driver transistors.**
- **The display can be blanked by BD (low) line.**
- **The display section consists of 16 x 8 display RAM. The CPU can read from or write into any location of the display RAM.**

Scan section:

- **The scan section has a scan counter and four scan lines, SL0 to SL3.**
- **In decoded scan mode, the output of scan lines will be similar to a 2-to-4 decoder.**
- **In encoded scan mode, the output of scan lines will be binary count, and so an external decoder should be used to convert the binary count to decoded output.**
- **The scan lines are common for keyboard and display.**
- **The scan lines are used to form the rows of a matrix keyboard and also connected to digit drivers of a multiplexed display, to turn ON/OFF.**

CPU interface section:

- **The CPU interface section takes care of data transfer between 8279 and the processor.**
- **This section has eight bidirectional data lines DB0 to DB7 for data transfer between 8279 and CPU.**
- **It requires two internal address A =0 for selecting data buffer and A = 1 for selecting control register of8279.**
- **The control signals WR (low), RD (low), CS (low) and A0 are used for read/write to 8279.**
- **It has an interrupt request line IRQ, for interrupt driven data transfer with processor.**
- **The 8279 require an internal clock frequency of 100 kHz. This can be obtained by dividing the input clock by an internal prescaler.**
- **The RESET signal sets the 8279 in 16-character display with two -key lockout keyboard modes.**

3.4.2 Keyboard Interface of 8279

The keyboard matrix can be any size from 2x2 to 8x8.Pins SL2-SL0 sequentially scan each column through a counting operation.The 74LS138 drives 0's on one line at a time.The 8279 scans RL pins synchronously with the scan.RL pins incorporate internal pull-ups, no need for external resistor pull-ups.The 8279 must be programmed first.

First three bits given below select one of 8 control registers (opcode).

- 000DDMMM

Mode set: Opcode 000.

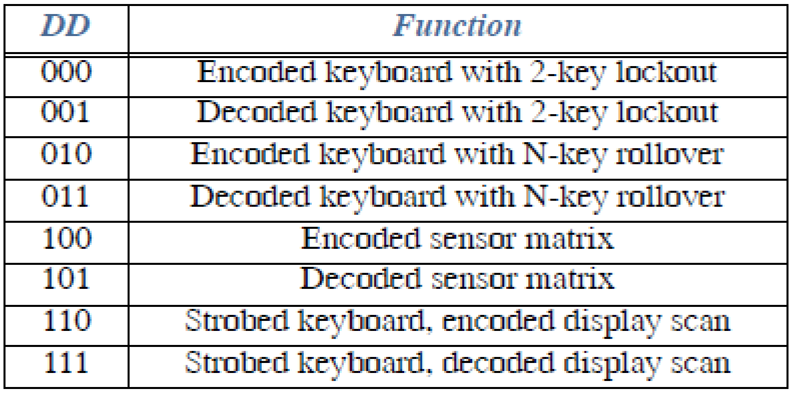
DD sets displays mode.

MMM sets keyboard mode.

DD field selects either:

- *8- or 16-digit display*
- *Whether new data are entered to the rightmost or leftmost display position.*

MMM field:



DD	*Function*
000	Encoded keyboard with 2-key lockout
001	Decoded keyboard with 2-key lockout
010	Encoded keyboard with N-key rollover
011	Decoded keyboard with N-key rollover
100	Encoded sensor matrix
101	Decoded sensor matrix
110	Strobed keyboard, encoded display scan
111	Strobed keyboard, decoded display scan

Encoded: *SL outputs are active-high, follow binary bit pattern 0-7 or 0-15.*

Decoded: *SL outputs are active-low (only one low at any time).*

Pattern output: 1110, 1101, 1011, 0111.

Strobed: *An active high pulse on the CN/ST input pin strobes data from the RL pins into an internal FIFO for reading by micro later.*

2-key lockout/N-key rollover: *Prevents 2 keys from being recognized if pressed simultaneously/Accepts all keys pressed from 1st to last.*

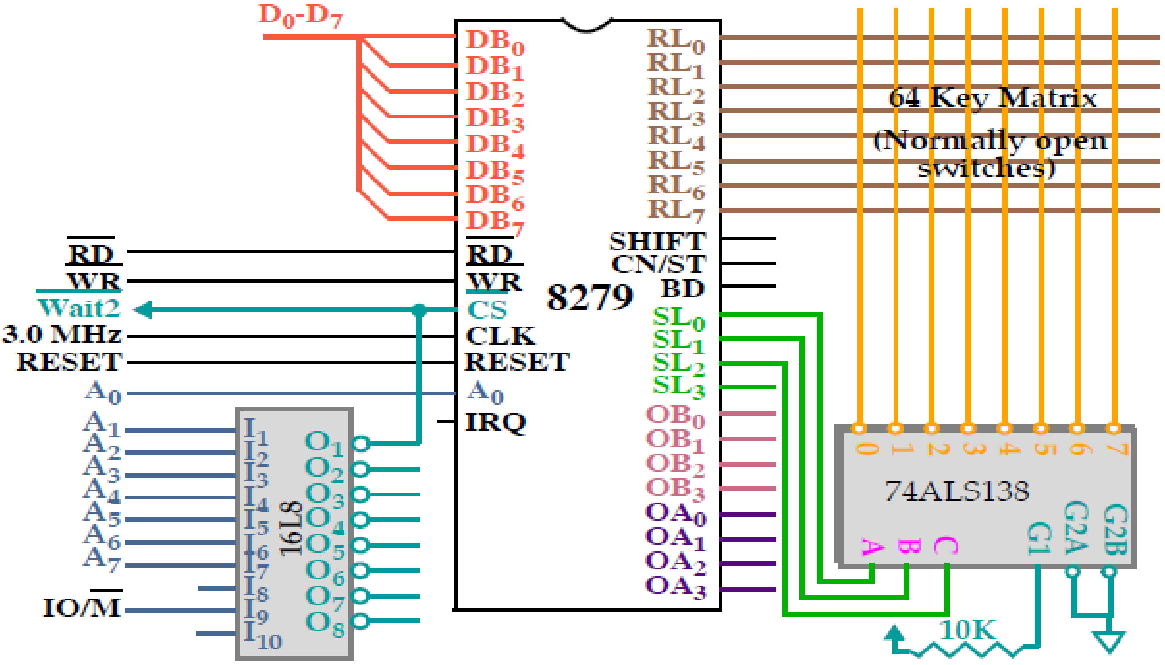


Fig 3.13 Keyboard Interface of 8279

Six Digit Display Interface of 8279

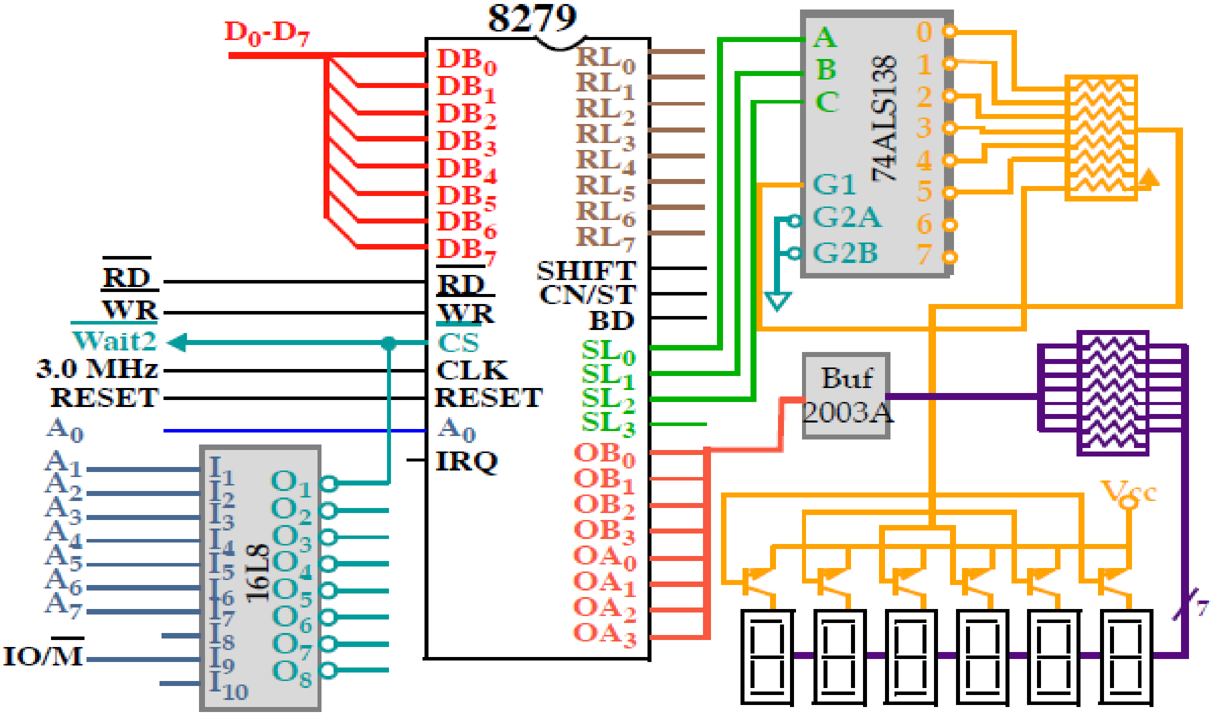
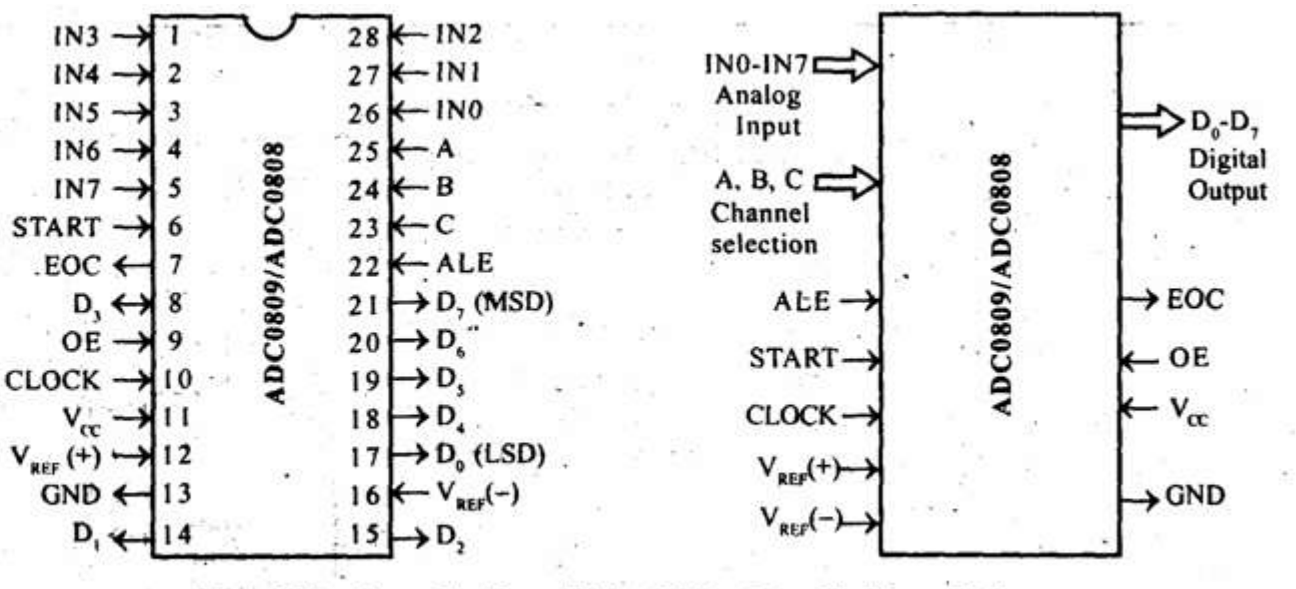


Fig 3.14 Display Interface of 8279

3.5 ADC Interfacing with 8085 Microprocessor

3.5.1 Features

- The ADC0809 is an 8-bit successive approximation type ADC with inbuilt 8-channel multiplexer.
- The ADC0809 is suitable for interface with 8086 microprocessor.
- The ADC0809 is available as a 28 pin IC in DIP (Dual Inline Package).
- The ADC0809 has a total unadjusted error of ±1 LSD (Least Significant Digit).
- The ADC0808 is also same as ADC0809 except the error. The total unadjusted error in ADC0808 is ± 1/2 LSD.



LSD = Least Significant Digit, MSD = Most Significant Digit

Fig 3.15 Pin Diagram of ADC 0809

3.5.2 Block Diagram of ADC 0809

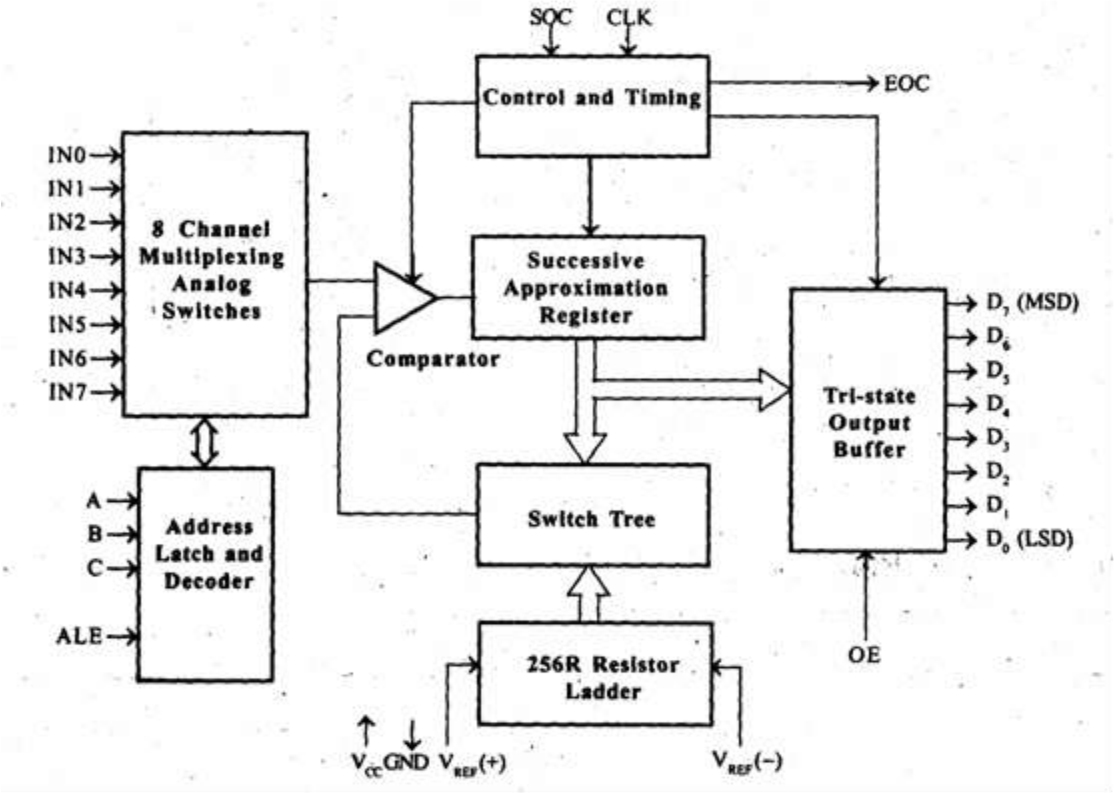
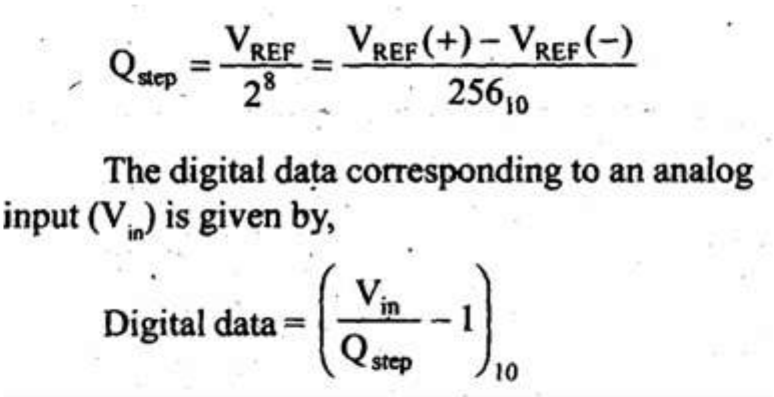


Fig 3.16 Block Diagram of ADC 0809

The successive approximation register (SAR) performs eight iterations to determine the digital code for input value. The SAR is reset on the positive edge of START pulse and start the conversion process on the falling edge of START pulse. A conversion process will be interrupted on receipt of new START pulse. The End-Of-Conversion (EOC) will go low between 0 and 8 clock pulses after the positive edge of START pulse. The ADC can be used in continuous conversion mode by tying the EOC output to START input. In this mode an external START pulse should be applied whenever power is switched ON.

The 256R ladder network has been provided instead of conventional R/2R ladder because of its inherent monotonic, which guarantees no missing digital codes. Also the 256R resistor network does not cause load variations on the reference voltage. The comparator in ADC0809/ADC0808 is a chopper- stabilized comparator. It converts the DC input signal into an AC signal, and amplifies the AC sign using high gain AC amplifier.

Then it converts AC signal to DC signal. This technique limits the drift component of the amplifier, because the drift is a DC component and it is not amplified/passed by the AC amp1ifier. This makes the ADC extremely insensitive to temperature, long term drift and input offset errors. In ADC conversion process the input analog value is quantized and each quantized analog value will have a unique binary equivalent. The quantization step in ADC0809/ADC0808 is given by,



$$Q_{step} = \frac{V_{REF}}{2^8} = \frac{V_{REF}(+) - V_{REF}(-)}{256_{10}}$$

The digital data corresponding to an analog input (V_{in}) is given by,

$$\text{Digital data} = \left(\frac{V_{in}}{Q_{step}} - 1 \right)_{10}$$

PROGRAM

ADDRESS	MNEMONICS	OPCODE	DESCRIPTION
	MVI A,10 OUT 0C8 H MVI A,18 OUT 0C8 H HLT		Channel 0 select ALE Low Channel 0, select ALE High

3.6 DAC Interfacing with 8085 Microprocessor

3.6.1 DAC 0800 Features

- To convert the digital signal to analog signal a Digital-to-Analog Converter (DAC) has to be employed.
- The DAC will accept a digital (binary) input and convert to analog voltage or current.
- Every DAC will have "n" input lines and an analog output.
- The DAC require a reference analog voltage (Vref) or current (Iref) source.
- The smallest possible analog value that can be represented by the n-bit binary code is called resolution.
- The resolution of DAC with n-bit binary input is 1/2nof reference analog value.

3.6.2 Circuit Diagram of DAC 0800

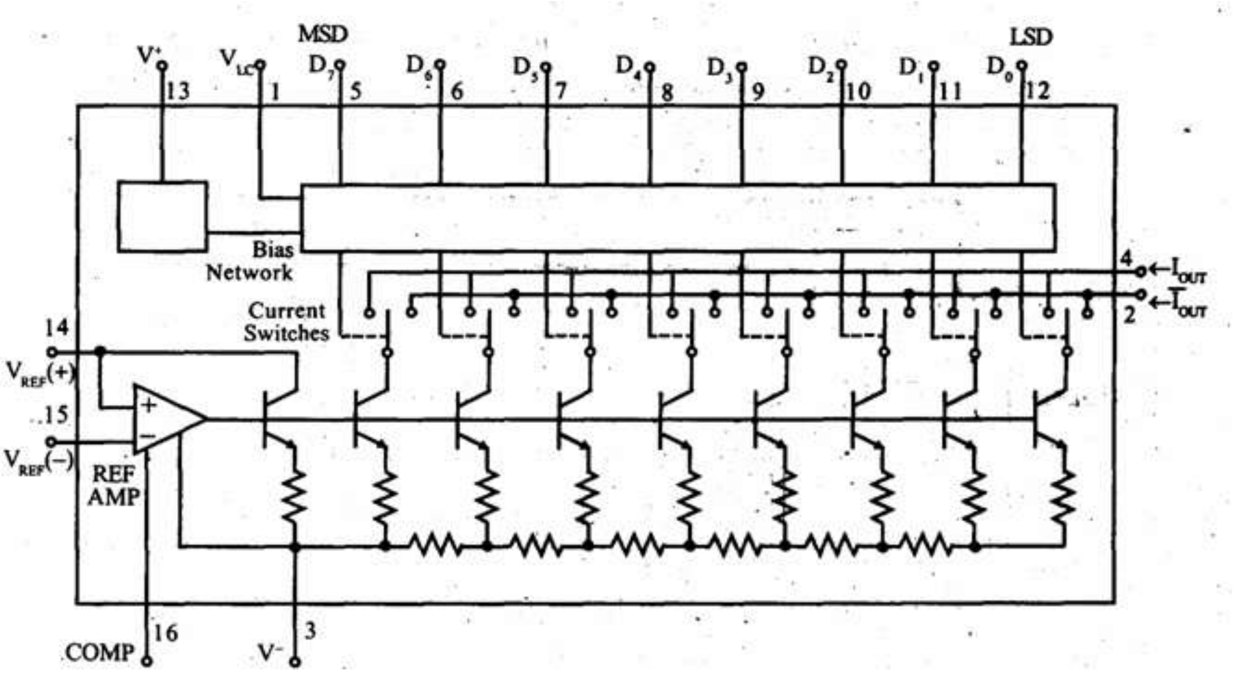


Fig 3.17 Circuit Diagram of DAC 0800

- The DAC0800 is an 8-bit, high speed, current output DAC with a typical settling time (conversion time) of 100 ns.

- It produces complementary current output, which can be converted to voltage by using simple resistor load.

- The DAC0800 require a positive and a negative supply voltage in the range of ± 5V to ±18V.

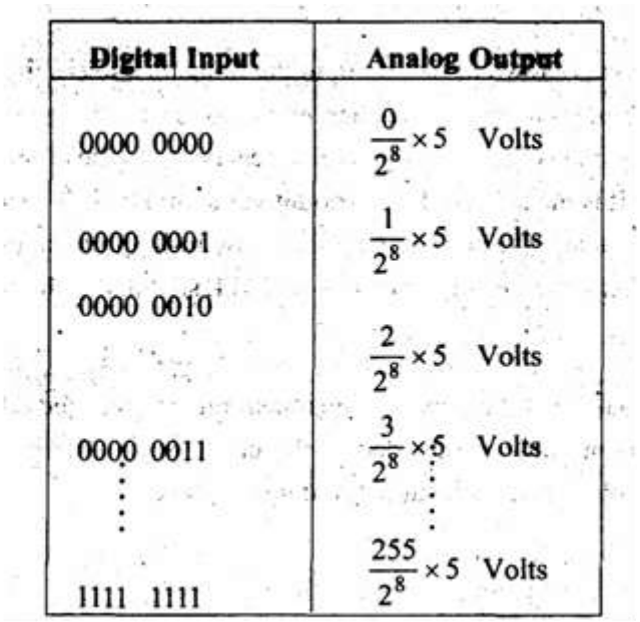
- It can be directly interfaced with TTL, CMOS, PMOS and other logic families.

- For TTL input, the threshold pin should be tied to ground (VLC = 0V).

- The reference voltage and the digital input will decide the analog output current, which can be converted to a voltage by simply connecting a resistor to output terminal or by using an op-amp I to V converter.

- The DAC0800 is available as a 16-pin IC in DIP.

Table 3.2 ADC Conversion Table



Digital Input	Analog Output
0000 0000	$\frac{0}{2^8} \times 5$ Volts
0000 0001	$\frac{1}{2^8} \times 5$ Volts
0000 0010	$\frac{2}{2^8} \times 5$ Volts
0000 0011	$\frac{3}{2^8} \times 5$ Volts
⋮	⋮
1111 1111	$\frac{255}{2^8} \times 5$ Volts

Square Wave Generation Using DAC 0800

ADDRESS	LABEL	MNEMONICS	OPCODE
	START	MVI A,00H	
		OUT C8	
		CALL DELAY	
		MVI A,FF	
		OUT C8	
		CALL DELAY	
		JMP START	
		MVI B,05H	
		MVI C,FF	
	DELAY	DCR C	
	L2	JNZ L1	
	L1	DCR B	
		JNL L2	

		RET	

UNIT IV

8051 MICRO CONTROLLER

4.1 Architecture of 8051:

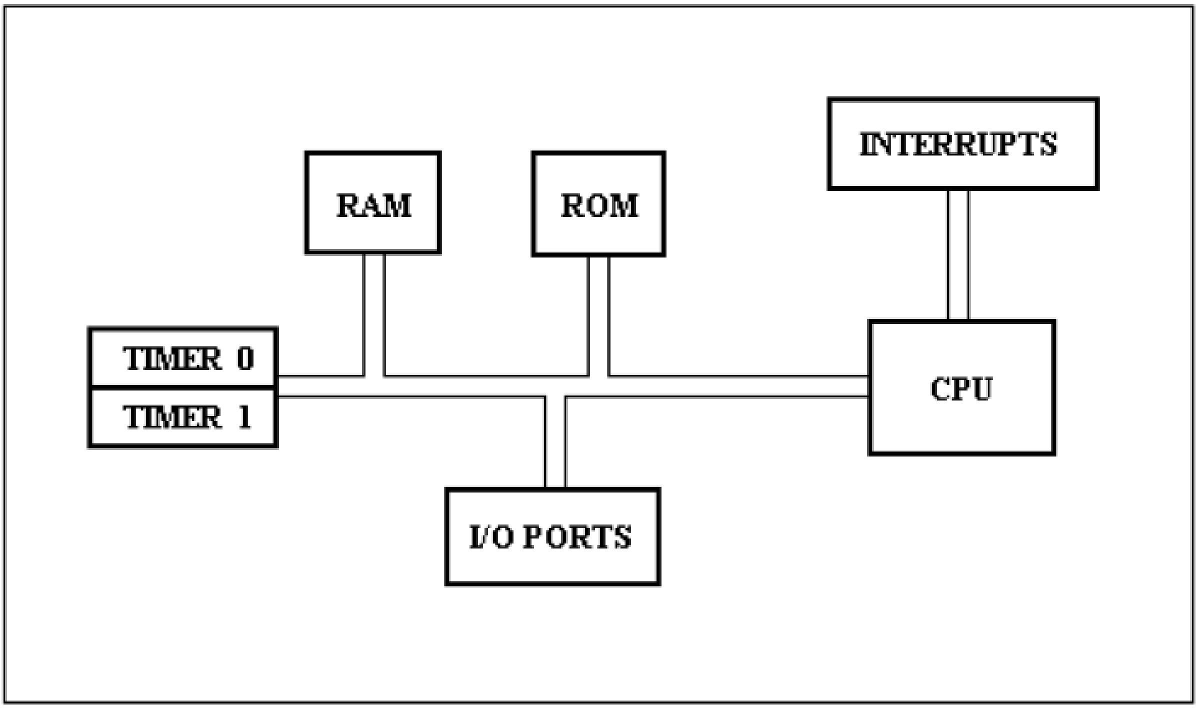


Fig 4.1 Architecture of 8051

4.1.1 Memory Organization

- Logical separation of program and data memory

-Separate address spaces for Program (ROM) and Data (RAM) Memory

-Allow Data Memory to be accessed by 8-bit addresses quickly and manipulated by

8-bit CPU

- Program Memory

  -Only be read, not written to

  -The address space is 16-bit, so maximum of 64K bytes

  -Up to 4K bytes can be on-chip (internal) of 8051 core

  -PSEN (Program Store Enable) is used for access to external Program Memory

  - Data Memory

  -Includes 128 bytes of on-chip Data Memory which are more easily accessible directly by its instructions

  -There is also a number of Special Function Registers (SFRs)

  -Internal Data Memory contains four banks of eight registers and a special 32-byte long segment which is bit addressable by 8051 bit-instructions

  -External memory of maximum 64K bytes is accessible by “movx”

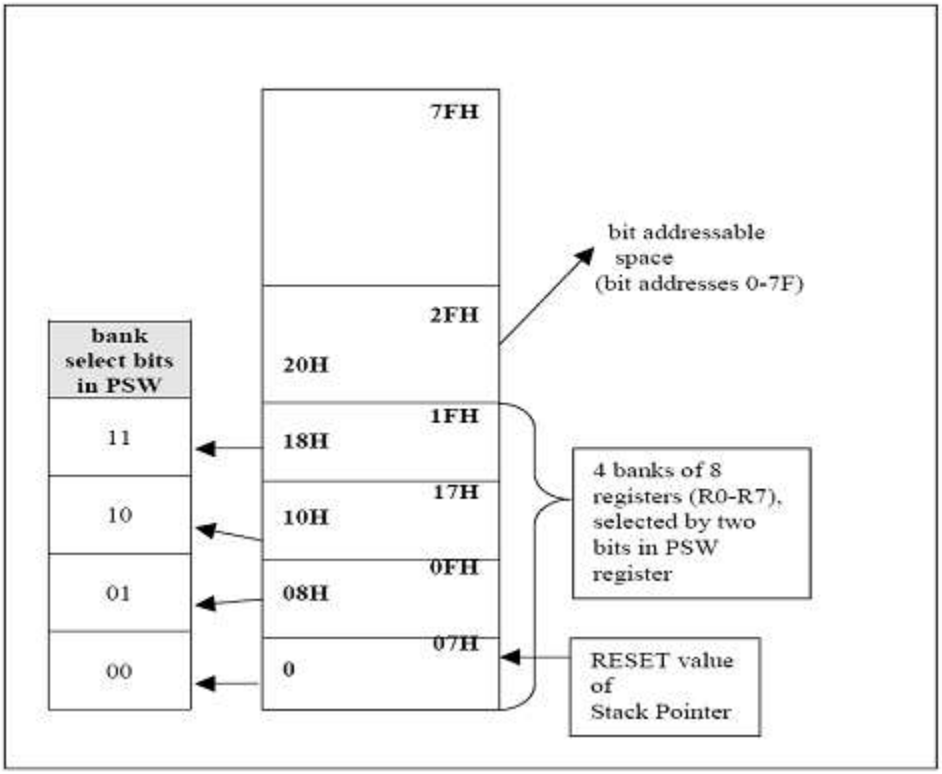

7FH
bit addressable space (bit addresses 0-7F)
2FH
20H
bank select bits in PSW
1FH
18H
11
17H
10H
10
4 banks of 8 registers (R0-R7), selected by two bits in PSW register
0FH
08H
01
07H
0
00
RESET value of Stack Pointer

Fig 4.2 Internal data Memory

4.1.2 Interrupt Structure

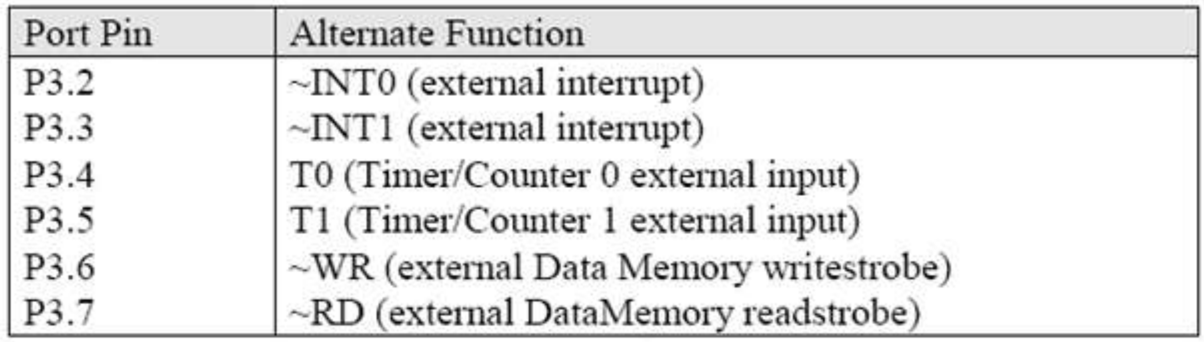
- The 8051 provides 4 interrupt sources
  - Two external interrupts
  - Two timer interrupts

4.1.3 Port Structure

- The 8051 contains four I/O ports
- All four ports are bidirectional
- Each port has SFR (Special Function Registers P0 through P3) which works like a latch, an output driver and an input buffer
- Both output driver and input buffer of Port 0 and output driver of Port 2 are used for accessing external memory
- Accessing external memory works like this

    - Port 0 outputs the low byte of external memory address (which is time-multiplexed with the byte being written or read)
    - Port 2 outputs the high byte (only needed when the address is 16 bits wide)
- Port 3 pins are multifunctional
- The alternate functions are activated with the 1 written in the corresponding bit in the port SFR

Table 4.1 Alternate Functions of Port 3 pins



Port Pin	Alternate Function
P3.2	~INT0 (external interrupt)
P3.3	~INT1 (external interrupt)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	~WR (external Data Memory writestrobe)
P3.7	~RD (external DataMemory readstrobe)

4.1.4 Timer/Counter

- The 8051 has two 16-bit Timer/Counter registers
    - Timer 0
    - Timer 1
- Both can work either as timers or event counters
- Both have four different operating modes

4.2 Instruction Format

An **instruction** is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts: one is task to be performed, called the **operation code** (opcode), and the second is the data to be operated on, called the **operand.** The operand (or data) can be specified in various ways. It may include 8-bit

(or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address.
In some instructions, the operand is implicit.

Instruction word size

The 8051 instruction set is classified into the following three groups according to word size:

1. One-word or 1-byte instructions

2. Two-word or 2-byte instructions

3. Three-word or 3-byte instructions

4.2.1 One-Byte Instructions

A 1-byte instruction includes the opcode and operand in the same byte. Operand(s) are internal register and are coded into the instruction.

These instructions are 1-byte instructions performing three different tasks. In the first instruction, both operand registers are specified. In the second instruction, the operand B is specified and the accumulator is assumed. Similarly, in the third instruction, the accumulator is assumed to be the implicit operand. These instructions are stored in 8-bit binary format in memory; each requires one memory location.

4.2.2 Two-Byte Instructions

In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode.

4.2.3 Three-Byte Instructions

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address.

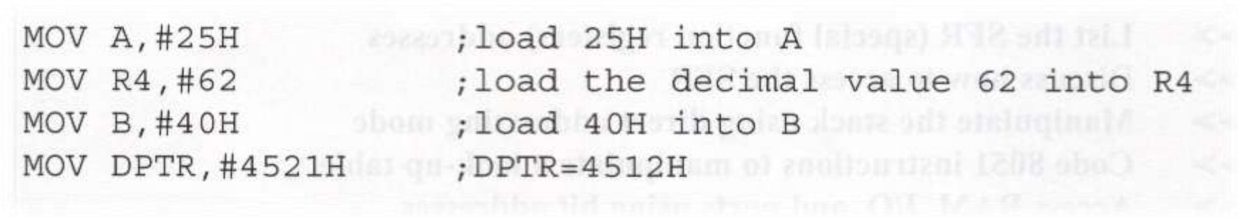
4.3 Addressing Modes of 8051

- **The 8051 provides a total of five distinct addressing modes.**
  - **(1) immediate**
  - **(2) register**
  - **(3) direct**

- **(4) register indirect**
- **(5) indexed**

(1) Immediate Addressing Mode

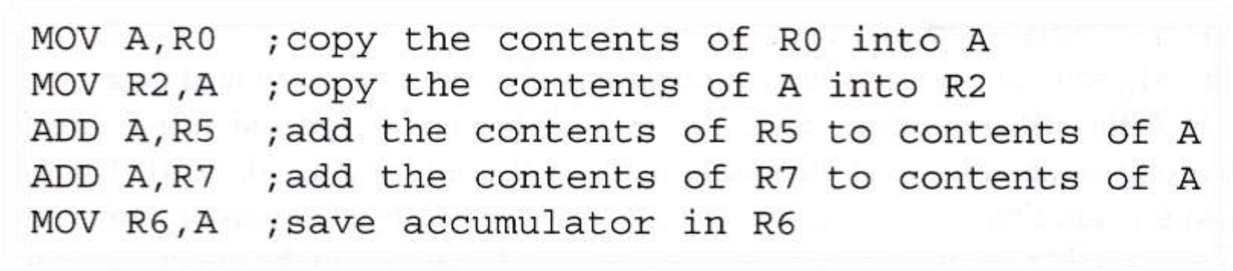
- The operand comes immediately after the op-code.
- The immediate data must be preceded by the pound sign, "#".



```
MOV A,#25H          ;load 25H into A
MOV R4,#62          ;load the decimal value 62 into R4
MOV B,#40H          ;load 40H into B
MOV DPTR,#4521H     ;DPTR=4512H
```

(2) Register Addressing Mode

- Register addressing mode involves the use of registers to hold the data to be manipulated



```
MOV A,R0   ;copy the contents of R0 into A
MOV R2,A   ;copy the contents of A into R2
ADD A,R5   ;add the contents of R5 to contents of A
ADD A,R7   ;add the contents of R7 to contents of A
MOV R6,A   ;save accumulator in R6
```

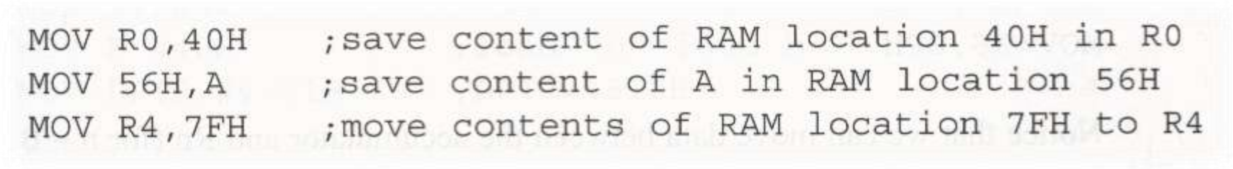
(3)Direct Addressing Mode

- It is most often used to access RAM locations 30 - 7FH.

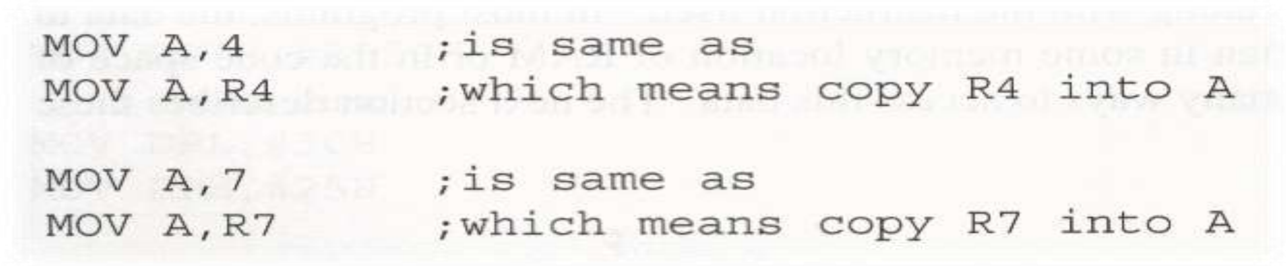
-This is due to the fact that register bank locations are accessed by the register names of R0 - R7.

-There is no such name for other RAM locations so must use direct addressing

-In the direct addressing mode, the data is in a RAM memory location whose address is known, and this address is given as a part of the instruction



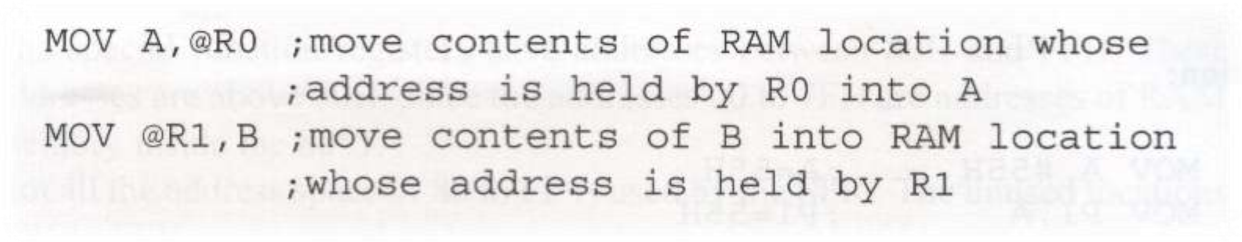
```
MOV R0,40H    ;save content of RAM location 40H in R0
MOV 56H,A     ;save content of A in RAM location 56H
MOV R4,7FH    ;move contents of RAM location 7FH to R4
```



```
MOV  A,4          ;is  same  as
MOV  A,R4         ;which  means  copy  R4  into  A

MOV  A,7          ;is  same  as
MOV  A,R7         ;which  means  copy  R7  into  A
```

***(4)Register Indirect* Addressing Mode**

- A register is used as a pointer to the data.
- If the data is inside the CPU, only registers R0 and R 1 are used for this purpose.
- R2 - R7 cannot be used to hold the address of an operand located in RAM when using indirect addressing mode.
- When RO and R 1 are used as pointers they must be preceded by the @ sign.



```
MOV A,@R0 ;move contents of RAM location whose
          ;address is held by R0 into A
MOV @R1,B ;move contents of B into RAM location
          ;whose address is held by R1
```

(5) Indexed Addressing Mode

- Indexed addressing mode is widely used in accessing data elements of look-up table entries located in the program ROM space of the 8051.
- The instruction used for this purpose is :

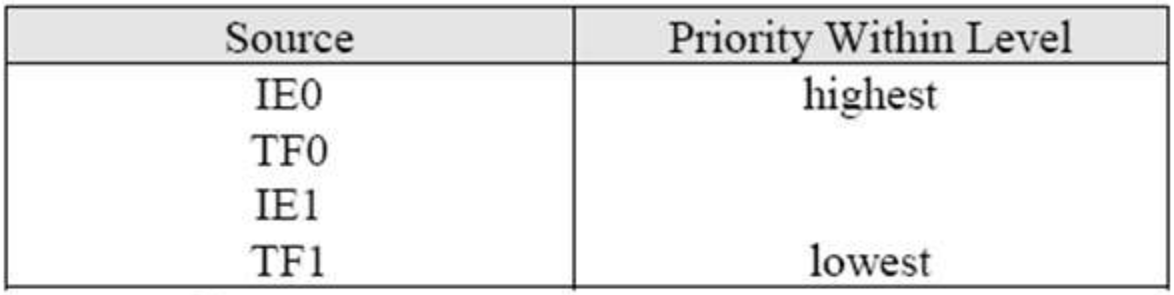
MOVC A, @ A+DPTR

- The 16-bit register DPTR and register A are used to form the address of the data element stored in on-chip ROM.
- Because the data elements are stored in the program (code) space ROM of the 8051, the instruction MOVC is used instead of MOV. The "C" means code.
- In this instruction the contents of A are added to the 16-bit register DPTR to form the 16-bit address of the needed data.

4.4 Interrupt Structure

- 8051 provides 4 interrupt sources
    - 2 external interrupts
    - 2 timer interrupts
- They are controlled via two SFRs, IE and IP
- Each interrupt source can be individually enabled or disabled by setting or clearing a bit in IE (Interrupt Enable). IE also exists a global disable bit, which can be cleared to disable all interrupts at once
- Each interrupt source can also be individually set to one of two priority levels by setting or clearing a bit in IP (Interrupt Priority)
- A low-priority interrupt can be interrupted by high-priority interrupt, but not by another low-priority one
- A high-priority interrupt can’t be interrupted by any other interrupt source
- If interrupt requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced, so within each priority lever there is a second priority structure
- This internal priority structure is determined by the polling sequence, shown in the following table

Table 4.2 Interrupt Priority Level



Source	Priority Within Level
IE0	highest
TF0	
IE1	
TF1	lowest

4.4.1 External Interrupt

- External interrupts ~INT0 and ~INT1 have two ways of activation
    - Level-activated
    - Transition-activated
- This depends on bits IT0 and IT1 in TCON
- The flags that actually generate these interrupts are bits IE0 and IE1 in TCON
- On-chip hardware clears that flag that generated an external interrupt when the service routine is vectored to, but only if the interrupt was transition-activated
- When the interrupt is level-activated, then the external requesting source is controlling the request flag, not the on-chip hardware

4.4.2 Handling Interrupt

- When interrupt occurs (or correctly, when the flag for an enabled interrupt is found to be set (1)), the interrupt system generates an LCALL to the appropriate location in Program Memory, unless some other conditions block the interrupt
- Several conditions can block an interrupt
    - An interrupt of equal or higher priority level is already in progress
    - The current (polling) cycle is not the final cycle in the execution of the instruction in progress
    - The instruction in progress is RETI or any write to IE or IP registers
    - If an interrupt flag is active but not being responded to for one of the above conditions, must be still active when the blocking condition is removed, or the denied interrupt will not be serviced

- Next step is saving the registers on stack. The hardware-generated LCALL causes only the contents of the Program Counter to be pushed onto the stack, and reloads the PC with the beginning address of the service routine

- In some cases it also clears the flag that generated the interrupt, and in other cases it doesn’t. It clears an external interrupt flag (IE0 or IE1) only if it was transition-avtivated.

- Having only PC be automatically saved gives programmer more freedom to decide how much time to spend saving other registers. Programmer must also be more careful with proper selection, which register to save.

- The service routine for each interrupt begins at a fixed location. The interrupt locations are spaced at 8-byte interval, beginning at 0003H for External Interrupt 0, 000BH for Timer 0, 0013H for External Interrupt 1 and 001BH for Timer 1.

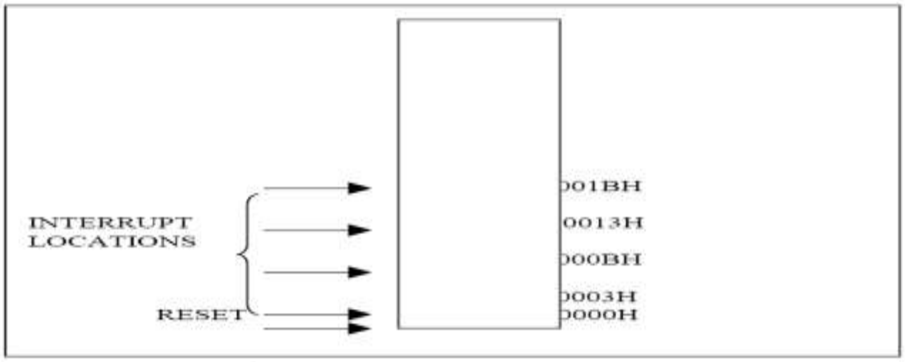


Fig 4.3 Interrupt Location in 8051 Program Memory

4.5 I/O Ports

- The 8051 contains four I/O ports

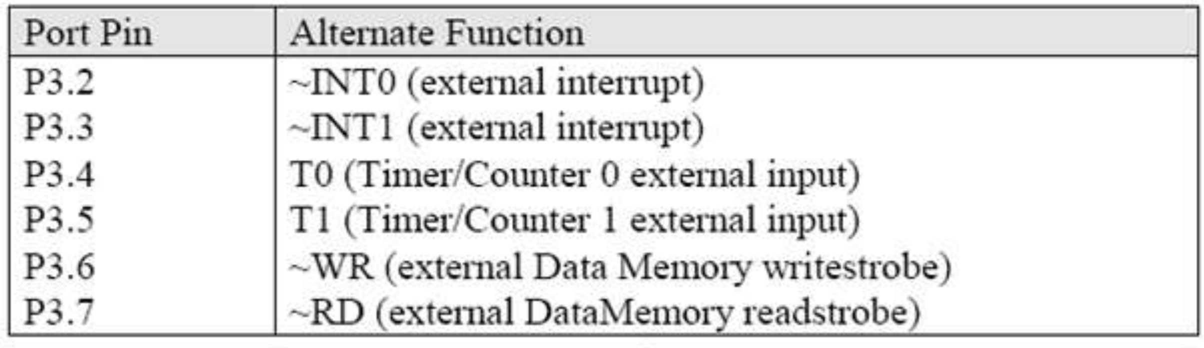
- All four ports are bidirectional

- Each port has SFR (Special Function Registers P0 through P3) which works like a latch, an output driver and an input buffer

- Both output driver and input buffer of Port 0 and output driver of Port 2 are used for accessing external memory

- Accessing external memory works like this
    - Port 0 outputs the low byte of external memory address (which is time-multiplexed with the byte being written or read)
    - Port 2 outputs the high byte (only needed when the address is 16 bits wide)
- Port 3 pins are multifunctional
- The alternate functions are activated with the 1 written in the corresponding bit in the port SFR

Table 4.3 Alternate Functions of Port 3 pins



Port Pin	Alternate Function
P3.2	~INT0 (external interrupt)
P3.3	~INT1 (external interrupt)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	~WR (external Data Memory writestrobe)
P3.7	~RD (external DataMemory readstrobe)

4.6 Timers

The 8051 comes equipped with two timers, both of which may be controlled, set, read, and configured individually. The 8051 timers have three general functions: 1) Keeping time and/or calculating the amount of time between events, 2) Counting the events themselves, or 3) Generating baud rates for the serial port.

one of the primary uses of timers is to measure time. We will discuss this use of timers first and will subsequently discuss the use of timers to count events. When a timer is used to measure time it is also called an "interval timer" since it is measuring the time of the interval between two events.

4.6.1Timer SFR

8051 has two timers which each function essentially the same way. One timer is TIMER0 and the other is TIMER1. The two timers share two SFRs (TMOD and TCON) which control the timers, and each timer also has two SFRs dedicated solely to itself (TH0/TL0 and TH1/TL1).

Table 4.4 SFR

SFR Name	Description	SFR Address
TH0	Timer 0 High Byte	8Ch
TL0	Timer 0 Low Byte	8Ah
TH1	Timer 1 High Byte	8Dh
TL1	Timer 1 Low Byte	8Bh
TCON	Timer Control	88h
TMOD	Timer Mode	89h

4.6.2 13-bit Time Mode (mode 0)

Timer mode "0" is a 13-bit timer. This is a relic that was kept around in the 8051 to maintain compatability with its predecesor, the 8048. Generally the 13-bit timer mode is not used in new development.

When the timer is in 13-bit mode, TLx will count from 0 to 31. When TLx is incremented from 31, it will "reset" to 0 and increment THx. Thus, effectively, only 13 bits of the two timer bytes are being used: bits 0-4 of TLx and bits 0-7 of THx. This also means, in essence, the timer can only contain 8192 values. If you set a 13-bit timer to 0, it will overflow back to zero 8192 machine cycles later.

Again, there is very little reason to use this mode and it is only mentioned so you wont be surprised if you ever end up analyzing archaeic code which has been passed down through the generations (a generation in a programming shop is often on the order of about 3 or 4 months).

4.6.3 16-bit Time Mode (mode 1)

Timer mode "1" is a 16-bit timer. This is a very commonly used mode. It functions just like 13-bit mode except that all 16 bits are used.

TLx is incremented from 0 to 255. When TLx is incremented from 255, it resets to 0 and causes THx to be incremented by 1. Since this is a full 16-bit timer, the timer may contain up to 65536 distinct values. If you set a 16-bit timer to 0, it will overflow back to 0 after 65,536 machine cycles.

4.6.4 8-bit Time Mode (mode 2)

Timer mode "2" is an 8-bit auto-reload mode. What is that, you may ask? Simple. When a timer is in mode 2, THx holds the "reload value" and TLx is the timer itself. Thus, TLx starts counting up. When TLx reaches 255 and is subsequently incremented, instead of resetting to 0 (as in the case of modes 0 and 1), it will be reset to the value stored in THx.

4.6.5 Split Timer Mode (mode 3)

Timer mode "3" is a split-timer mode. When Timer 0 is placed in mode 3, it essentially becomes two separate 8-bit timers. That is to say, Timer 0 is TL0 and Timer 1 is TH0. Both timers count from 0 to 255 and overflow back to 0. All the bits that are related to Timer 1 will now be tied to TH0.

While Timer 0 is in split mode, the real Timer 1 (i.e. TH1 and TL1) can be put into modes 0, 1 or 2 normally--however, you may not start or stop the real timer 1 since the bits that do that are now linked to TH0. The real timer 1, in this case, will be incremented every machine cycle no matter what.

4.6.6 USING TIMERS AS EVENT COUNTERS

We've discussed how a timer can be used for the obvious purpose of keeping track of time. However, the 8051 also allows us to use the timers to count events.

How can this be useful? Let's say you had a sensor placed across a road that would send a pulse every time a car passed over it. This could be used to determine the volume of traffic on the road. We could attach this sensor to one of the 8051's I/O lines and constantly monitor it, detecting when it pulsed high and then incrementing our counter when it went back to a low state. This is not terribly difficult, but requires some code. Let's say we hooked the sensor to P1.0; the code to count cars passing would look something like this:

```
JNB P1.0,$        ;If a car hasn't raised the signal, keep waiting

JB P1.0,$         ;The line is high which means the car is on the sensor right now

INC COUNTER ;The car has passed completely, so we count it
```

4.7 Serial Communication

Some of the external I/0 devices receive only the serial data.Normally serial communication is used in the Multi Processor environment.8051 has two pins for serial communication.

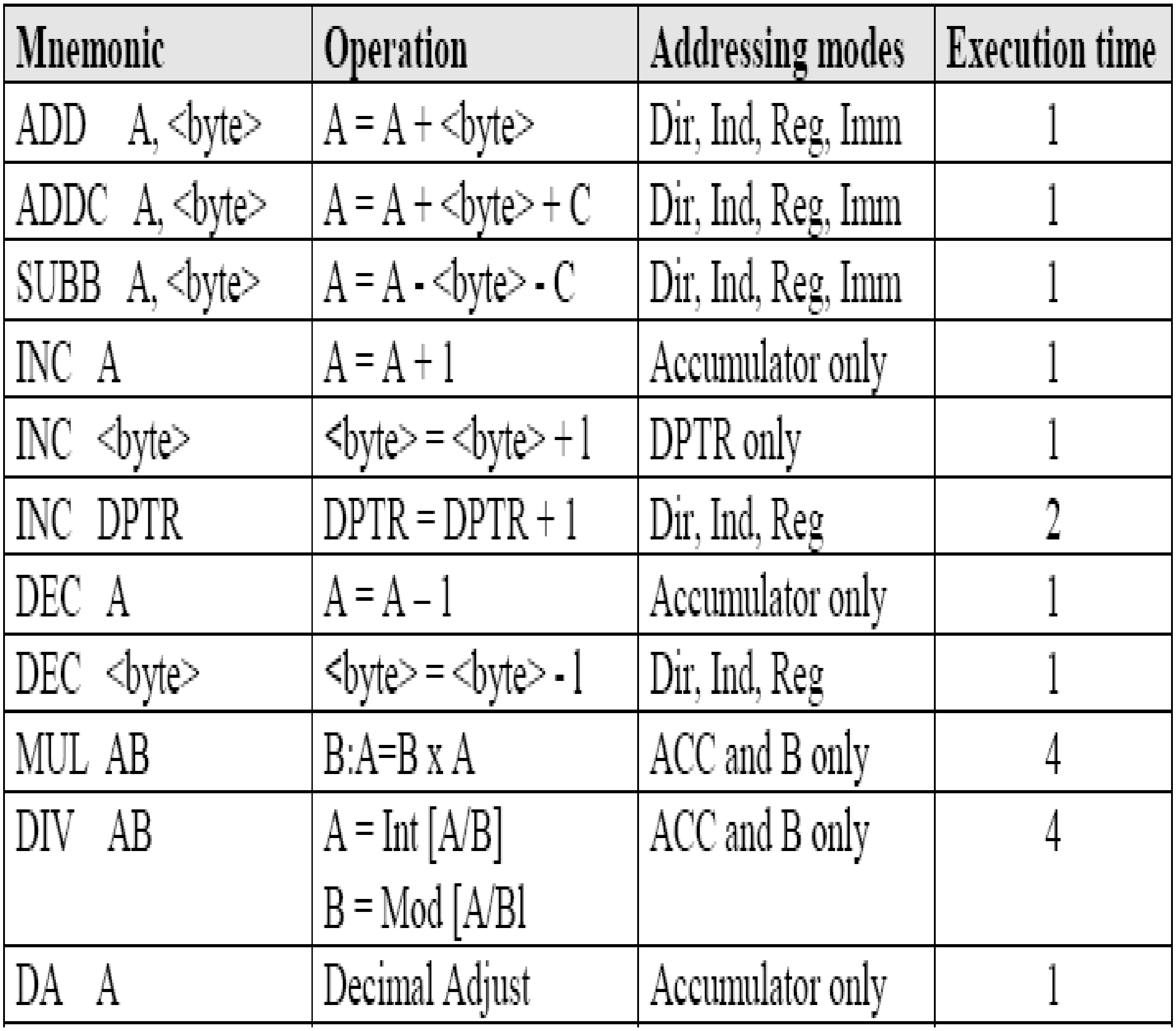
(1)SID- Serial Input data.

(2)SOD-Serial Output data.

UNIT V

MICRO CONTROLLER PROGRAMMING & APPLICATIONS

5.1 Arithmetic Instructions

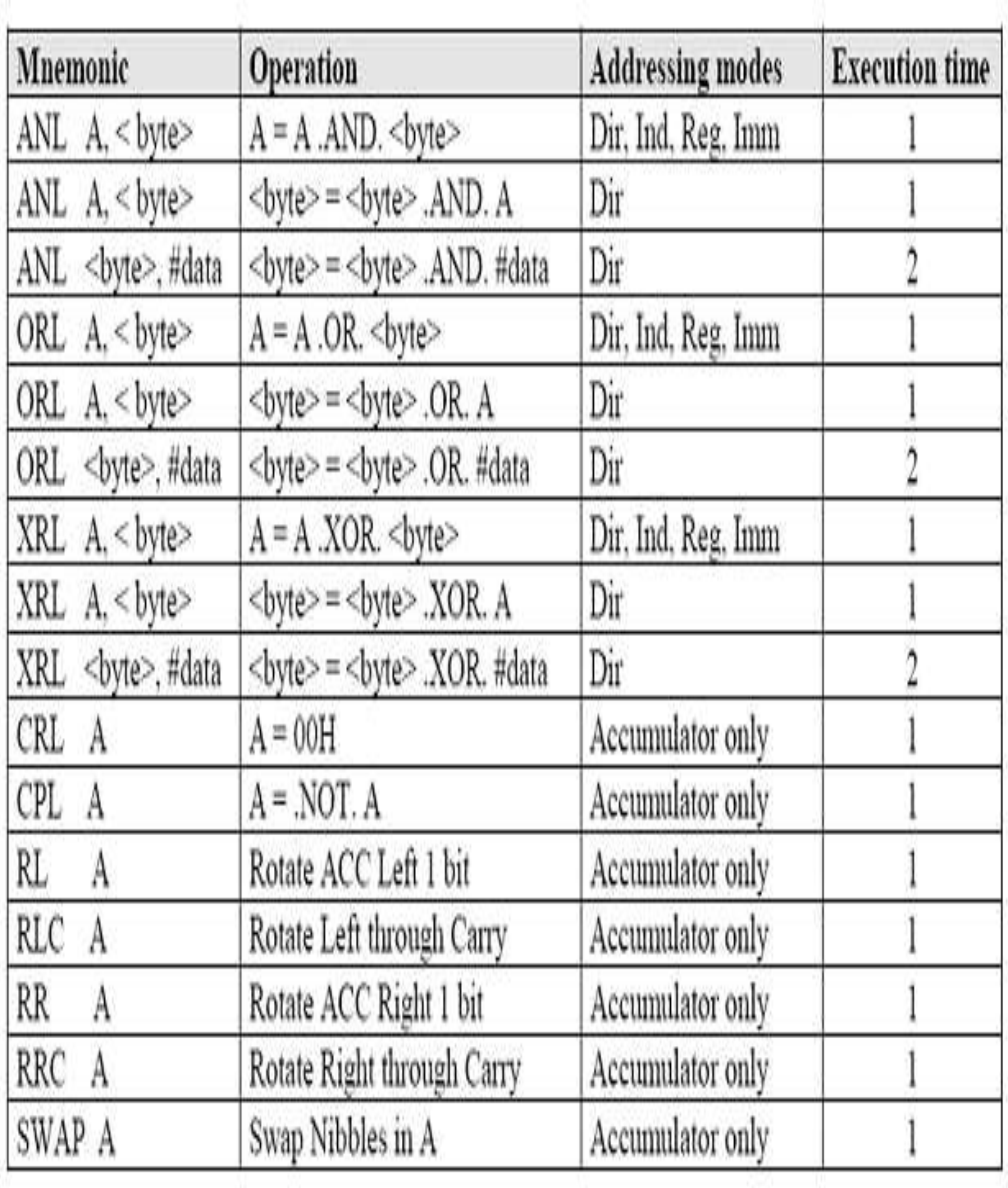


Mnemonic	Operation	Addressing modes	Execution time
ADD A, <byte>	A = A + <byte>	Dir, Ind, Reg, Imm	1
ADDC A, <byte>	A = A + <byte> + C	Dir, Ind, Reg, Imm	1
SUBB A, <byte>	A = A - <byte> - C	Dir, Ind, Reg, Imm	1
INC A	A = A + 1	Accumulator only	1
INC <byte>	<byte> = <byte> + 1	DPTR only	1
INC DPTR	DPTR = DPTR + 1	Dir, Ind, Reg	2
DEC A	A = A – 1	Accumulator only	1
DEC <byte>	<byte> = <byte> - 1	Dir, Ind, Reg	1
MUL AB	B:A=B x A	ACC and B only	4

DIV AB	A = Int [A/B]
	B = Mod [A/Bl
		ACC and B only	4

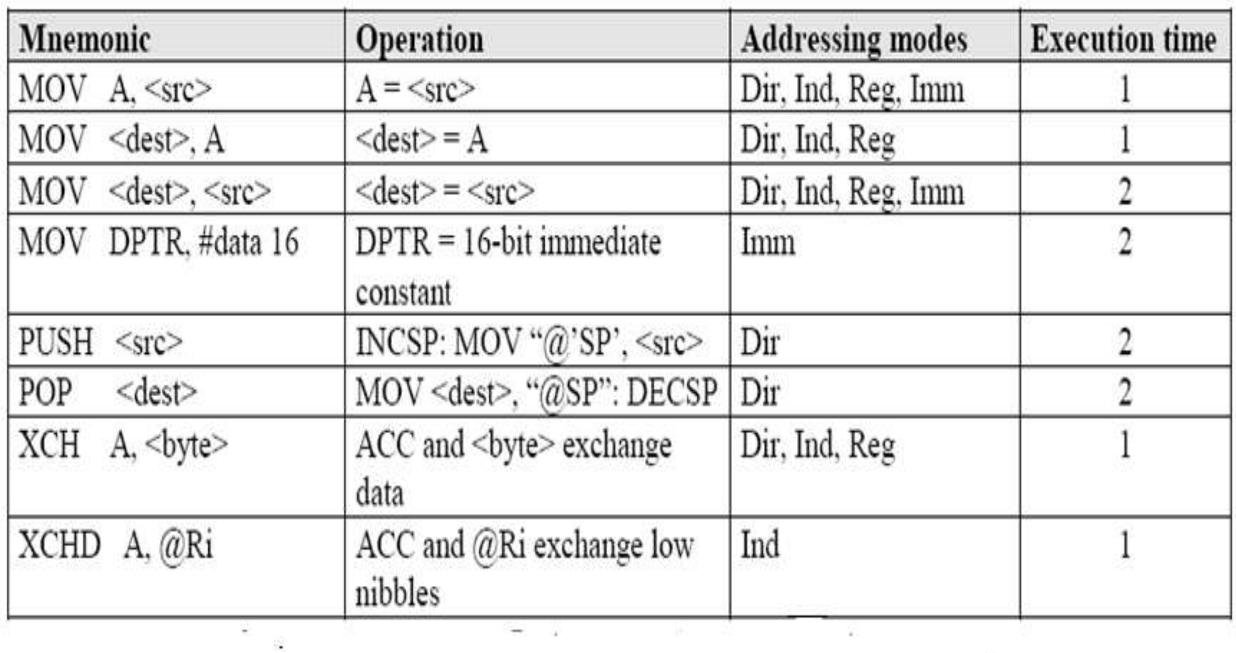
DA A	Decimal Adjust	Accumulator only	1

5.2 Logical Instructions



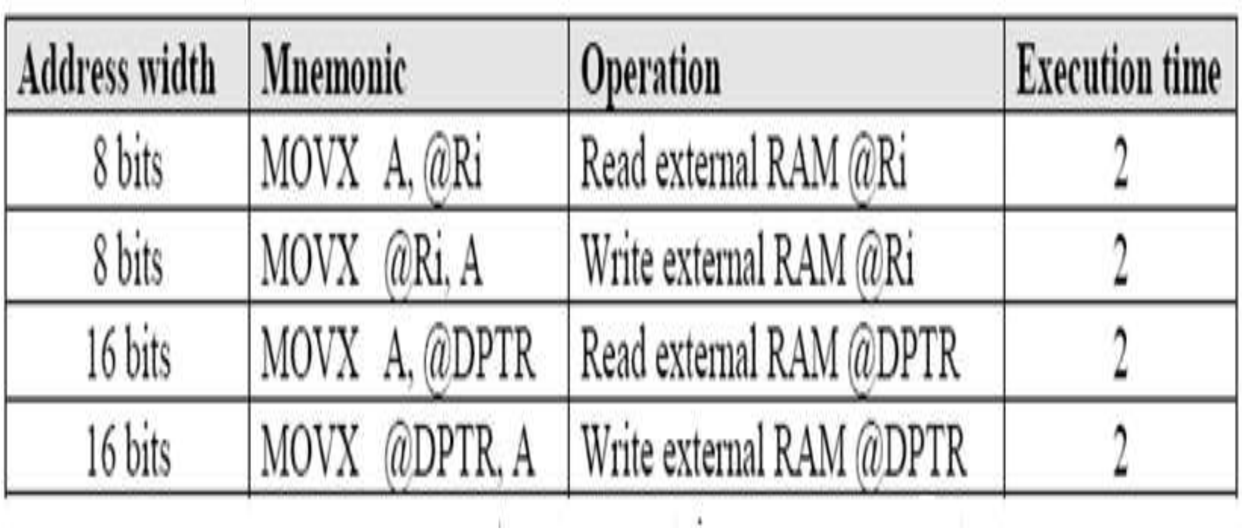
Mnemonic	Operation	Addressing modes	Execution time
ANL A, < byte>	A = A .AND. <byte>	Dir, Ind, Reg, Imm	1
ANL A, < byte>	<byte> = <byte> .AND. A	Dir	1
ANL <byte>, #data	<byte> = <byte> .AND. #data	Dir	2
ORL A, < byte>	A = A .OR. <byte>	Dir, Ind, Reg, Imm	1
ORL A, < byte>	<byte> = <byte> .OR. A	Dir	1
ORL <byte>, #data	<byte> = <byte> .OR. #data	Dir	2
XRL A, < byte>	A = A .XOR. <byte>	Dir, Ind, Reg, Imm	1
XRL A, < byte>	<byte> = <byte> .XOR. A	Dir	1
XRL <byte>, #data	<byte> = <byte> .XOR. #data	Dir	2
CRL A	A = 00H	Accumulator only	1
CPL A	A = .NOT. A	Accumulator only	1
RL A	Rotate ACC Left 1 bit	Accumulator only	1
RLC A	Rotate Left through Carry	Accumulator only	1
RR A	Rotate ACC Right 1 bit	Accumulator only	1
RRC A	Rotate Right through Carry	Accumulator only	1
SWAP A	Swap Nibbles in A	Accumulator only	1

5.3 Data Transfer Instructions that access the Internal Data Memory



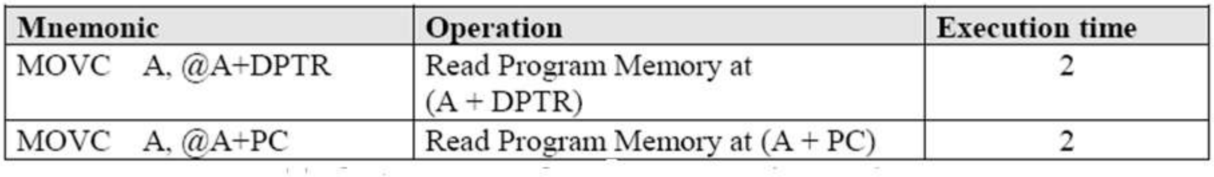
Mnemonic	Operation	Addressing modes	Execution time
MOV A, <src>	A = <src>	Dir, Ind, Reg, Imm	1
MOV <dest>, A	<dest> = A	Dir, Ind, Reg	1
MOV <dest>, <src>	<dest> = <src>	Dir, Ind, Reg, Imm	2
MOV DPTR, #data 16	DPTR = 16-bit immediate constant	Imm	2
PUSH <src>	INCSP: MOV "@'SP', <src>	Dir	2
POP <dest>	MOV <dest>, "@SP": DECSP	Dir	2
XCH A, <byte>	ACC and <byte> exchange data	Dir, Ind, Reg	1
XCHD A, @Ri	ACC and @Ri exchange low nibbles	Ind	1

5.4 Data Transfer Instructions that access the External Data Memory



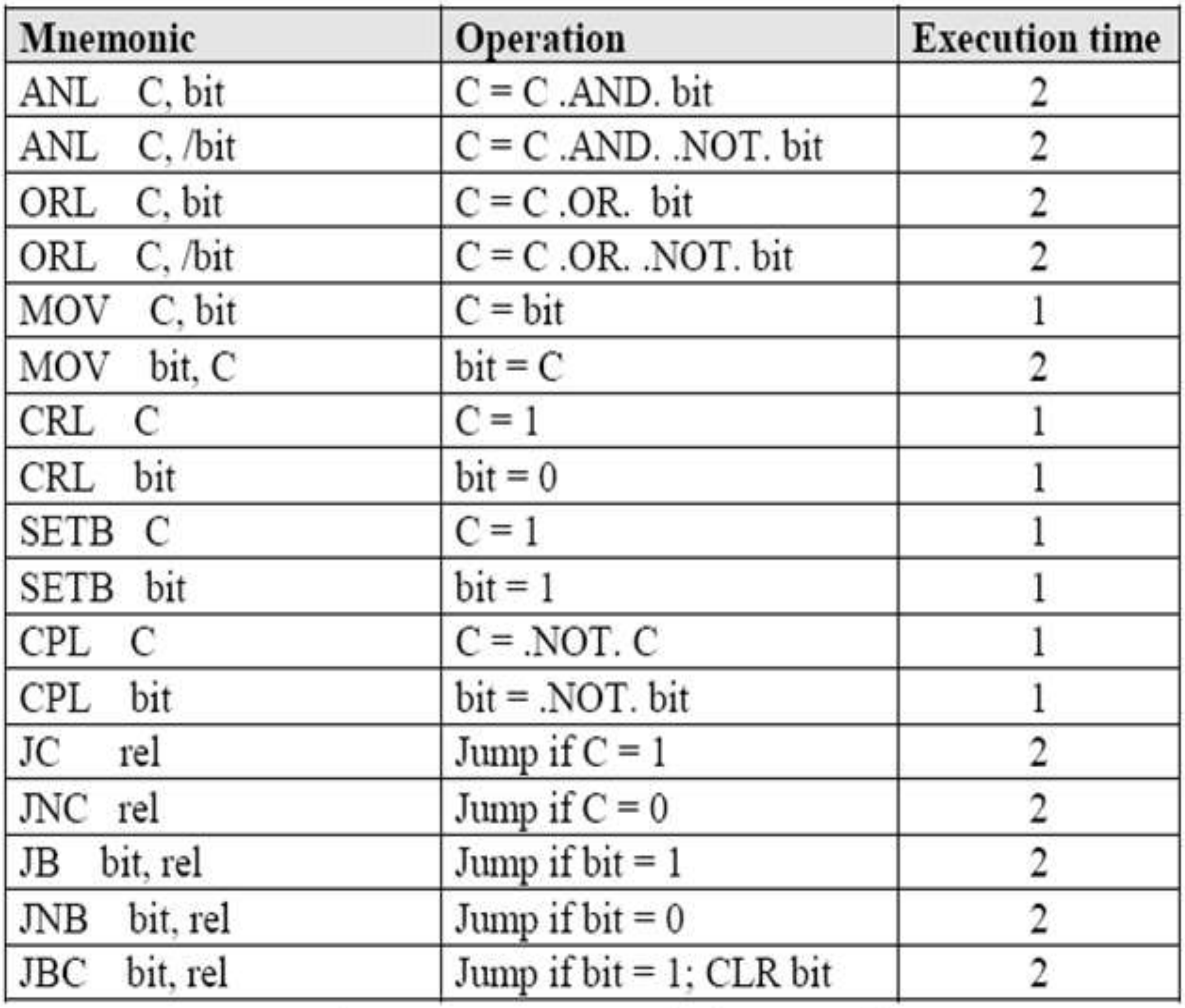
Address width	Mnemonic	Operation	Execution time
8 bits	MOVX A, @Ri	Read external RAM @Ri	2
8 bits	MOVX @Ri, A	Write external RAM @Ri	2
16 bits	MOVX A, @DPTR	Read external RAM @DPTR	2
16 bits	MOVX @DPTR, A	Write external RAM @DPTR	2

5.5 Look up Tables



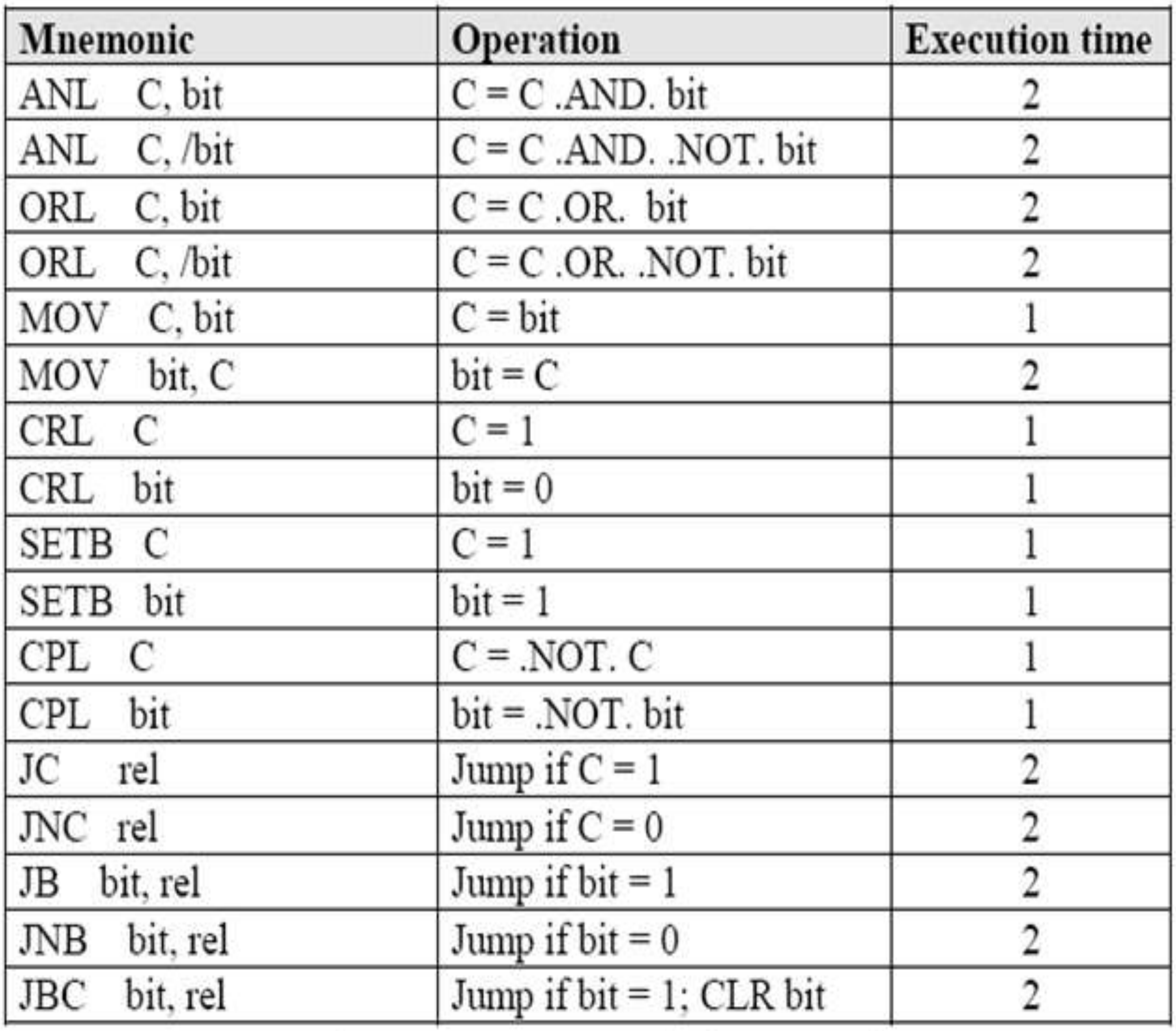
Mnemonic	Operation	Execution time
MOVC A, @A+DPTR	Read Program Memory at (A + DPTR)	2
MOVC A, @A+PC	Read Program Memory at (A + PC)	2

5.6 Boolean Instructions



Mnemonic	Operation	Execution time
ANL C, bit	C = C .AND. bit	2
ANL C, /bit	C = C .AND. .NOT. bit	2
ORL C, bit	C = C .OR. bit	2
ORL C, /bit	C = C .OR. .NOT. bit	2
MOV C, bit	C = bit	1
MOV bit, C	bit = C	2
CRL C	C = 1	1
CRL bit	bit = 0	1
SETB C	C = 1	1
SETB bit	bit = 1	1
CPL C	C = .NOT. C	1
CPL bit	bit = .NOT. bit	1
JC rel	Jump if C = 1	2
JNC rel	Jump if C = 0	2
JB bit, rel	Jump if bit = 1	2
JNB bit, rel	Jump if bit = 0	2
JBC bit, rel	Jump if bit = 1; CLR bit	2

5.7 Jump Instructions



Mnemonic	Operation	Execution time
ANL C, bit	C = C .AND. bit	2
ANL C, /bit	C = C .AND. .NOT. bit	2
ORL C, bit	C = C .OR. bit	2
ORL C, /bit	C = C .OR. .NOT. bit	2
MOV C, bit	C = bit	1
MOV bit, C	bit = C	2
CRL C	C = 1	1
CRL bit	bit = 0	1
SETB C	C = 1	1
SETB bit	bit = 1	1
CPL C	C = .NOT. C	1
CPL bit	bit = .NOT. bit	1
JC rel	Jump if C = 1	2
JNC rel	Jump if C = 0	2
JB bit, rel	Jump if bit = 1	2
JNB bit, rel	Jump if bit = 0	2
JBC bit, rel	Jump if bit = 1; CLR bit	2

5.8 Interfacing Keyboard to 8051 Microcontroller

The key board here we are interfacing is a matrix keyboard. This key board is designed with a particular rows and columns. These rows and columns are connected to the microcontroller through its ports of the micro controller 8051. We normally use 8*8 matrix key board. So only two ports of 8051 can be easily connected to the rows and columns of the key board.

When ever a key is pressed, a row and a column gets shorted through that pressed key and all the other keys are left open. When a key is pressed only a bit in the port goes high. Which indicates microcontroller that the key is pressed. By this high on the bit key in the corresponding column is identified.

Once we are sure that one of key in the key board is pressed next our aim is to identify that key. To do this we firstly check for particular row and then we check the corresponding column the key board.

To check the row of the pressed key in the keyboard, one of the row is made high by making one of bit in the output port of 8051 high . This is done until the row is found out. Once we get the row next out job is to find out the column of the pressed key. The column is detected by contents in the input ports with the help of a counter. The content of the input port is rotated with carry until the carry bit is set.

The contents of the counter is then compared and displayed in the display. This display is designed using a seven segment display and a BCD to seven segment decoder IC 7447.

The BCD equivalent number of counter is sent through output part of 8051 displays the number of pressed key.

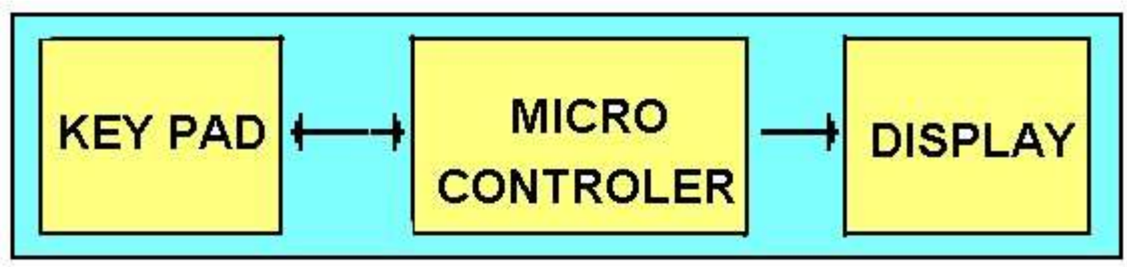


Fig 5.1 Interfacing Keyboard to 8051 Microcontroller

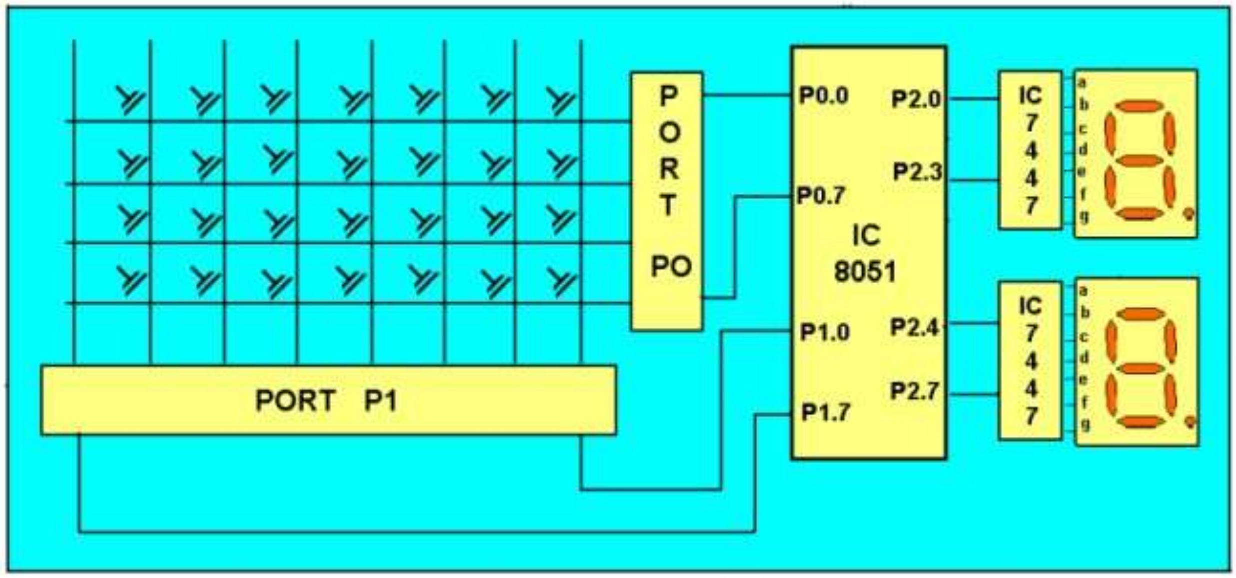


Fig 5.2 Circuit Diagram of Interfacing Keyboard to 8051

5.9 Program for Keyboard Interfacing with 8051

Start of main program:

to check that whether any key is pressed

```
start:   mov a,#00h
         mov p1,a              ;making all rows of port p1 zero
         mov a,#0fh
         mov p1,a              ;making all rows of port p1 high
press:   mov a,p2
         jz press              ;check until any key is pressed
```

after making sure that any key is pressed

```
         mov a,#01h            ;make one row high at a time
         mov r4,a
         mov r3,#00h            ;initiating counter
next:    mov a,r4
         mov p1,a              ;making one row high at a time
         mov a,p2              ;taking input from port A
         jnz colscan           ;after getting the row jump to check
                                  column
         mov a,r4
         rl a                  ;rotate left to check next row
         mov r4,a
         mov a,r3
         add a,#08h           ;increment counter by 08 count
         mov r3,a
         sjmp next            ;jump to check next row
```

after identifying the row to check the colomn following steps are followed

```
colscan:   mov r5,#00h
     in:   rrc a          ;rotate right with carry until get the carry
           jc out         ;jump on getting carry
           inc r3         ;increment one count
           jmp in
    out:   mov a,r3
           da a           ;decimal adjust the contents of counter
                               before display
           mov p2,a
           jmp start      ;repeat for check next key.
```

5.10 Seven Segment Disply Interfacing with 8051

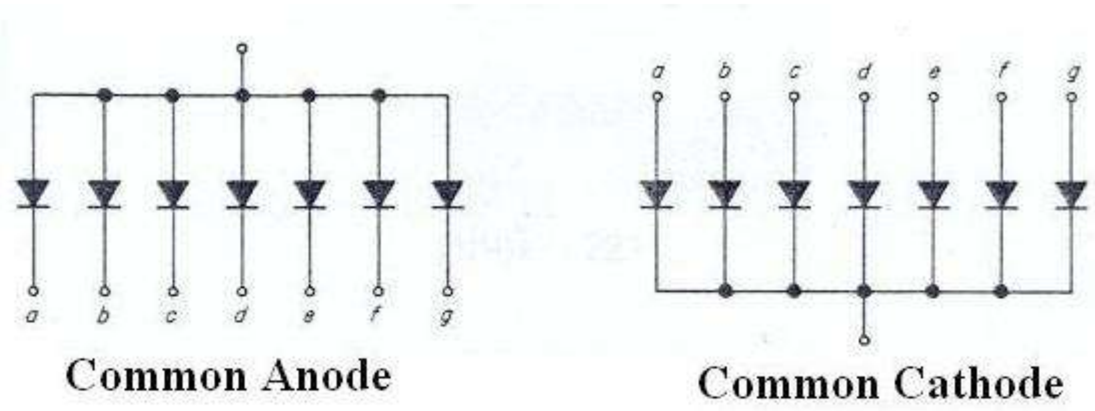


Fig 5.3 Interfacing LEDS to 8051 Microcontroller

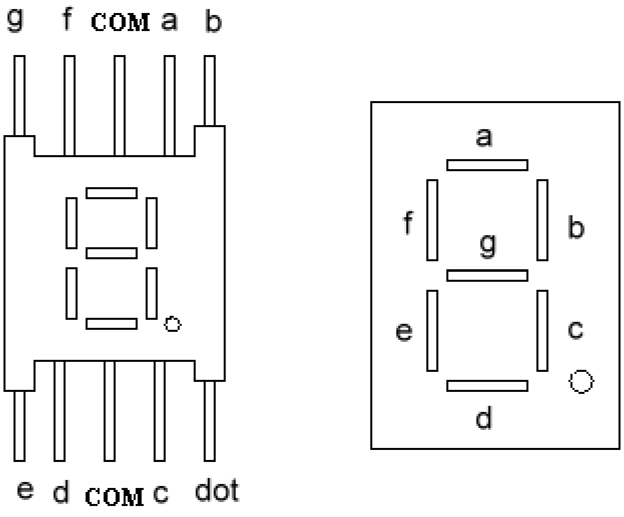


Fig 5.4 Seven Segment Display

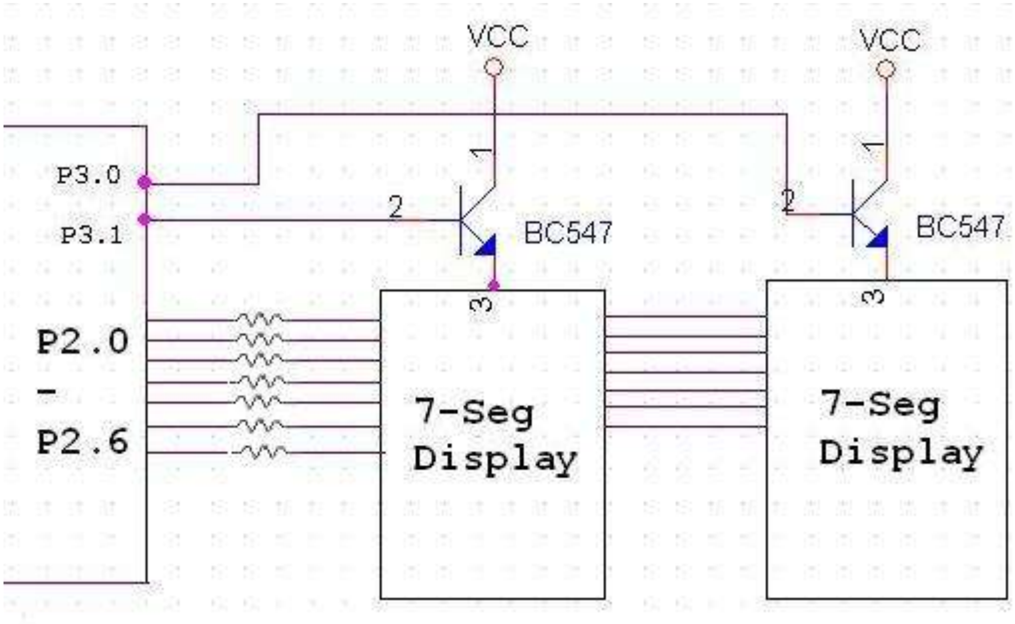


Fig 5.5 Connecting Seven segment Display with 8051

5.11 SEVEN SEGMENT COMMON ANODE DISPLAY CONNECTED TO PORT2

```
ZERO EQU 0C0H

ONEEQU 0F9H

TWOEQU 0A4H

THREE EQU 0B0H

FOUREQU 99H

FIVEEQU 92H

IXEQU 82H

SEVENEQU 0F8H

EIGHTEQU 80H

NINEEQU 90H

DOT EQU 7FH


ORG 00H
```

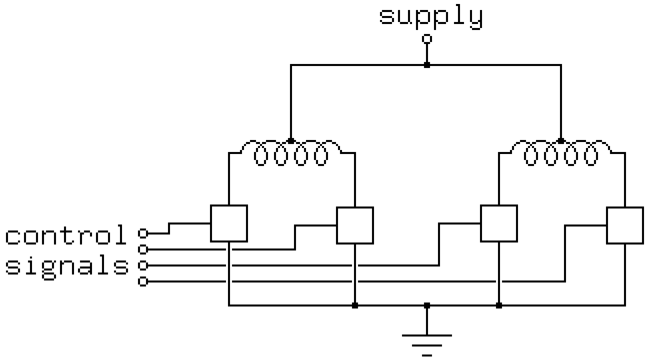
```
MOVP2,#00H
LOOP:
MOV P2,#ZERO
CALL DELAYS
MOV P2,#ONE
CALL DELAYS
MOV P2,#TWO
CALL DELAYS
MOV P2,#THREE
CALL DELAYS
MOV P2,#FOUR
CALL DELAYS
MOV P2,#FIVE
CALL DELAYS
MOV P2,#SIX
CALL DELAYS
MOV P2,#SEVEN
CALL DELAYS
MOV P2,#EIGHT
CALL DELAYS
MOV P2,#NINE
 CALL DELAYS

   MOV P2,#DOT
   CALL DELAYS
```

```
  AJMP LOOP


DELAYS ;1s DELAY

MOV R5,#10

D1:

CALL DELAY

DJNZ R5,D1

RET


DELAY:          ;100ms DELAY

MOV R7,#200D2:


MOV R6,#100

D3:

NOP

NOP

NOP

DJNZ R6,D3

DJNZ R7,D2

RET

END
```

5.12 Interfacing Stepper Motor with 8051Microcontroller

Step motor is the easiest to control. It's handling simplicity is really hard to deny - all there is to do is to bring the sequence of rectangle impulses to one input of step controller and direction information to another input. Direction information is very simple and comes down to "left" for logical one on that pin and "right" for logical zero. Motor control is also very simple -

every impulse makes the motor operating for one step and if there is no impulse the motor won't start. Pause between impulses can be shorter or longer and it defines revolution rate. This rate cannot be infinite because the motor won't be able to "catch up" with all the impulses (documentation on specific motor should contain such information). The picture below represents the scheme for connecting the step motor to microcontroller and appropriate program code follows.



The key to driving a stepper is realizing how the motor is constructed. A diagram shows the representation of a 4 coil motor, so named because 4 coils are used to cause the revolution of the drive shaft. Each coil must be energized in the correct order for the motor to spin.

5.12.1 Step angle

It is angle through which motor shaft rotates in one step. step angle is different for different motor . selection of motor according to step angle depends on the application , simply if you require small increments in rottion choose motor having smaller step angle.

No of steps require to rotate one complete rotation = 360 deg. / step angle in deg.

5.12.2 INTERFACING TO 8051.

To cause the stepper to rotate, we have to send a pulse to each coil in turn. The 8051 does not have sufficient drive capability on its output to drive each coil, so there are a number of ways to drive a stepper,

Stepper motors are usually controlled by transistor or driver IC like ULN2003.

Driving current for each coil is then needed about 60mA at +5V supply. A Darlington

```
         +5v
          |    _________    +------------------+---  +12v
+-------+-+   |        10|---+     __________    |
| 8051 1|-----|1 2803 18|--------|          |  |
|       2|-----|2      17|--------| stepper  |  |
|       3|-----|3      16|--------|          |--+
|       4|-----|4      15|--------|__________|  |
|        |     |         |                      |
|        |     |         |         __________   |
|       5|-----|5      14|--------|          |  |
|       6|-----|6      13|--------| stepper  |--+
|       7|-----|7      12|--------|          |
|       8|-----|8      11|--------|__________|
|        |  +--|9        |
+---+---+  |  +----------+
    |      |
    +------+-----gnd
```

transistor array, ULN2003 is used to increase driving capacity of the 2051 chip. Four 4.7k resistors help the 2051 to provide more sourcing current from the +5V supply.

Table 5.1

Coil A	Coil B	Coil C	Coil D	Step
0	1	1	0	1
0	0	1	1	2
1	0	0	1	3
1	1	0	0	4

5.12.3 CODE EXAMPLE

To move motor in forward direction continuously

Connection -P1.0 -P1.3 connected to Coils A -D.

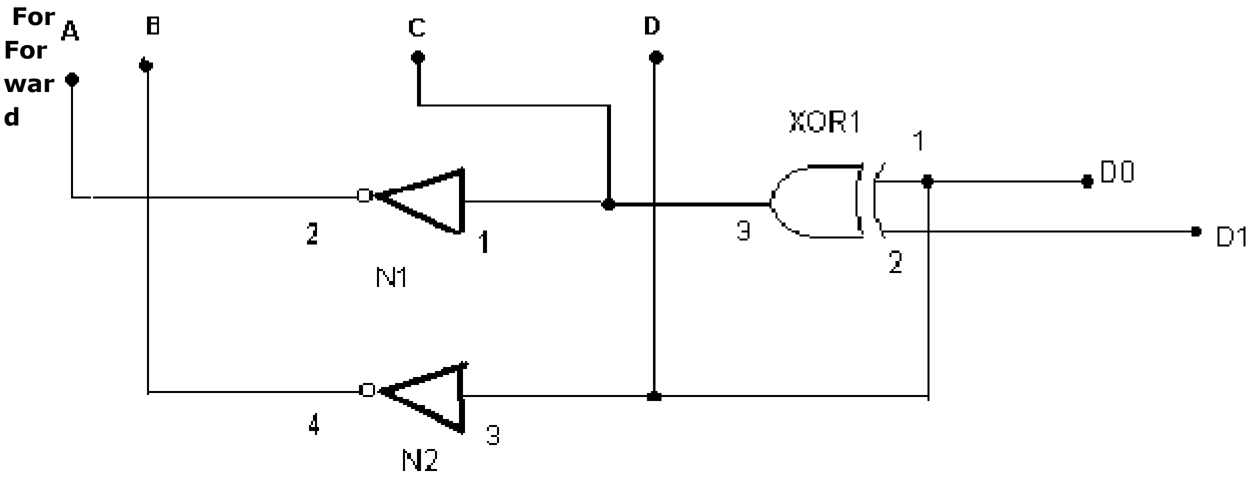
EMBLY LANGUAGE

```
mov a,#66h  ;Load step sequence


AGAIN
mov p2,a ;issue sequence to motor

r a       ;rotate step sequence right clockwise=Next sequence

call DELAY  ;~ 20 msec.

jmp AGAIN   ;Repete again
```

C LANGUAGE   (SPJ)

```
void main ()
{
TMOD = 0x20 ;
TCON = 0x40 ;
TH1 = 0xf9 ;
TL1 = 0xf9 ;
PCON = 0x80 ;
SCON = 0x50 ;
while (1) /*continues loop */
{
printf("a"); /* transmit a along with CR & LF.
}
```

5.12.4 CONTROLLING STEPPER MOTOR WITH TWO PORT PINS ONLY



D0	D0	Coil energized
0	0	AB
0	1	BC
1	0	CD
1	1	DA

5.12.5 CODE:

```
// controlling a stepper motor

#include <stdio.h> //
#include <reg420.h> //
#include <ctype.h> //
#include "serial.h"

void main()
{
char o;
int i;
InitSerialHardware();

do{
```

```
o = getchar();

if(isspace(o)) continue;
o = toupper(o);

if(o == 'S')
{
puts(" Stop");
P1 = 0;
}

if (o == 'L')
{
puts(" Left");
{
TMOD = 0x20;
TCON = 0x40;
TH1 = 0xF9;
TL1 = 0xF9;
PCON = 0x80;
SCON0 = 0x50;
}
}

if (o == 'R')
{
puts(" Right");
{
SCON0 = 0x50;
PCON = 0x80;
TL1 = 0xF9;
TH1 = 0xF9;
TCON = 0x40;
TMOD = 0x20;
}
}else continue;

}while (1); /*continues loop */

{
printf("a"); // transmit a along with CR & LF.
}
}
```

5.13 Servo Motor

Servos are DC motors with built in gearing and feedback control loop circuitry. And no motor drivers required. They are extremely popular with robot, RC plane, and RC boat builders. Most servo motors can rotate about 90 to 180 degrees. Some rotate through a full 360degreesormore.

However, servos are unable to continually rotate, meaning they can't be used for driving wheels, unless they are modified (how to modify), but their precision positioning makes them ideal for robot legs and arms, rack and pinion steering, and sensor scanners to name a few. Since servos are fully self contained, the velocity and angle control loops are very easy to impliment, while prices remain very affordable. To use a servo, simply connect the black wire to ground, the red to a 4.8-6V source, and the yellow/white wire to a signal generator (such as from your microcontroller). Vary the square wave pulse width from 1-2 ms and yourservoisnowposition/velocitycontrolled.

Pulse width modulation (PWM) is a powerful technique for controlling analog circuits with a processor's digital outputs. PWM is employed in a wide variety of applications, ranging from measurement and communications to power control and conversion. The general concept is to simply send an ordinary logic square wave to your servo at a specific wave length, and your servo goes to a particular angle (or velocity if your servo is modified).Thewavelengthdirectlymapstoservoangle.

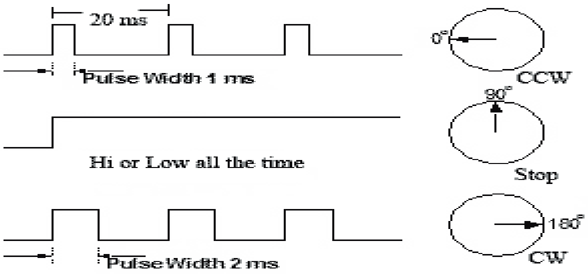


Fig 5.9 Pulse for controlling Servo motor

5.13.1 Controlling the Servo Motor

- **PWM**

Pulse width modulation (PWM) is a powerful technique for controlling analog circuits with a processor's digital outputs. PWM is employed in a wide variety of applications, ranging from measurement and communications to power control and conversion. The general concept is to simply send an ordinary logic square wave to your servo at a specific wave length, and your servo goes to a particular angle (or velocity if your servo is modified). The wavelength directly maps to servo angle.

- **Programmable Counter Array (PCA)**

The PCA is a special modules in Philips P89V51RD2 which includes a special 16-bit Timer that has five 16-bit capture/compare modules associated with it. Each of the modules can be programmed to operate in one of four modes: rising and/or falling edge capture, software timer, high-speed output, or pulse width modulator. Each module has a pin associated with it in port 1.

Module 0 is connected to P1.3 (CEX0), module 1 to P1.4 (CEX1), etc. Registers CH and CL contain current value of the free running up counting 16-bit PCA timer. The PCA timer is a common time base for all five modules and can be programmed to run at: 1/6 the oscillator frequency, 1/2 the oscillator frequency, the Timer 0 overflow, or the input on the ECI pin (P1.2). The timer count source is determined from the CPS1 and CPS0 bits in the CMOD SFR.

In the CMOD SFR there are three additional bits associated with the PCA. They are CIDL which allows the PCA to stop during idle mode, WDTE which enables or disables the Watchdog function on module 4, and ECF which when set causes an interrupt and the PCA overflow flag CF (in the CCON SFR) to be set when the PCA timer overflows. The Watchdog timer function is implemented in module 4 of PCA. Here, we are interested only PWM mode.

- **8051 Pulse width modulator mode**

All of the PCA modules can be used as PWM outputs. Output frequency depends on the source for the PCA timer. All of the modules will have the same frequency of output because they all share one and only PCA timer. The duty cycle of each module is independently variable using the module's capture register CCAPnL.When the value of the PCA CL SFR is less than the value in the module's CCAPnL SFR the output will be low, when it is equal to or greater than the output will be high. When CL overflows from FF to 00, CCAPnL is reloaded with the value in CCAPnH. this allows updating the PWM without glitches. The PWM and ECOM bits in the module's CCAPMn register must be set to enable the PWM mode. For more details see P89V51RD2 datasheet.

This is an example how to control servos with 8051 by using PWM. The schematic is shown below. I use P1.4 (CEX1) to control the left servo and P1.2 (CEX2) to control the right servo. Here, I use GWS servo motor model S03T STD. I need three states of duty cycle:

- 20 ms to Stop the servo
- 1 ms to Rotate Clockwise

Calculation for duty cycle (for XTAL 18.432 MHz with 6 Clock/Machine cycle)

- Initial PWM Period = 20mS (18.432MHz /6-Cycle Mode)
- Initial PCA Count From Timer0 Overflow
- 1 Cycle of Timer0 = (1/18.432MHz)x6 = 0.326 uS
- Timer0 AutoReload = 240 Cycle = 78.125 uS
- 1 Cycle PCA = [(1/18.432MHz)x6]x240 = 78.125 uS
- Period 20mS of PCA = 20ms/78.125us = 256 (CL Reload)
- CL (20mS) = 256 Cycle Auto Reload
- Load CCAPxH (1.0mS) = 256-13 = 243 (243,244,...,255 = 13 Cycle)
- Load CCAPxH (2.0mS) = 255-26 = 230 (230,231,...,255 = 26 Cycle)
- 2 ms to Rotate Counter-clockwise

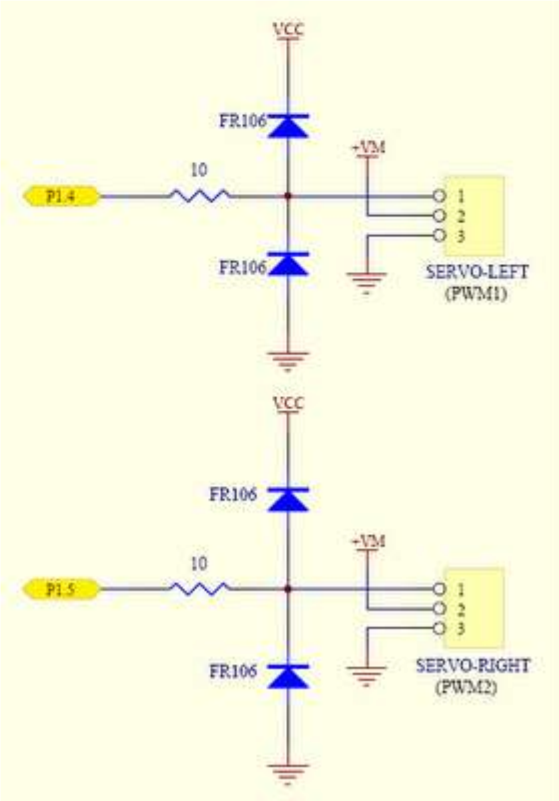


Fig 5.10 Schematic Control

5.13.2 Program

```
Filename : pwm_servos.h
 * Hardware : Controller  -> P89V51RD2
 *           XTAL        -> 18.432 MHz
 *           Mode       -> 6 Clock/MC
 * I/O      : P1.4       -> Left  (PWM-CEX1)
 *           P1.5       -> Right (PWM-CEX2)
 * Compiler : SDCC


/* Control the Left servo */
void ServoL_back()
{
   CCAP1H = 243;
}

void ServoL_forward()
{
   CCAP1H = 230;
}

void ServoL_stop()
{
   CCAP1H = 0;
}


/* Control the Right servo */
void ServoR_back()
{
   CCAP2H = 230;
}

void ServoR_forward()
{
   CCAP2H = 243;
}

void ServoR_stop()
{
   CCAP2H = 0;
}


/* Initialize the PCA and PWM mode */
void Servos_init()
{
/* Initial Timer0 Generate Overflow PCA */

   TMOD = 0x02;   /* Timer0 Mode2 : 8bit auto reload */
   TH0  = 16;     /* 256-240, 8.125usec Auto-relead (20msec/PWM) */
   TL0  = TH0;
   TCON = 0x10;   /* setb TR0, TCON or 0001,0000*/

/*
   Initial PWM Period  = 20mS (18.432MHz /6-Cycle Mode)
   Initial PCA Count From Timer0 Overflow
   1 Cycle of Timer0 = (1/18.432MHz)x6 = 0.326uS
   Timer0 AutoReload = 240 Cycle = 78.125uS
   1 Cycle PCA = [(1/18.432MHz)x6]x240 = 78.125uS
   Period 20mS of PCA = 20ms/78.125us = 256(CL Reload)
   CL(20mS) = 256 Cycle Auto Reload
   Load CCAPxH(1.0mS) = 256-13 = 243 (243,244,...,255 = 13 Cycle)
   Load CCAPxH(2.0mS) = 255-26 = 230 (230,231,...,255 = 26 Cycle)
```

```
*/
    CMOD=0x04;
    CCAPM1=0x42;
    CCAPM2=0x42;
    CCAP1H=0x00;
    CCAP2H=0x00;
    CCON=0x40;
}
```

test.c

```
#include <p89v51rd2.h>
#include "pwm_servos.h"

void PowerOn()
{
    unsigned char inner, outer;

    IE = 0x00;
    P1 = 0xFF; /* Motor STOP */

    for (outer = 0x00; outer < 0x10; outer++) { /* Delay for a while */
        for (inner = 0x00; inner < 0xFF; inner++);
    }

    Servos_init();

    IE = 0x80; /* Start interrupt */
}

void main()
{
    PowerOn();

    ServoR_forward();
    ServoL_back();

    while (1);
}
```

5.14 Washing Machine Control

Many washing m/c shell in the market has mechanical controlled sequence for activated the timer and the sequence back and forth for their motor; washing motor or spinning motor. Spinning motor control only has one direction only, and its simple could be changed to the discrete mechanical timer which sell on the market. But washing motor control has 2 direction for this purpose, it means to squeeze the clothes, it must go to forward and then reversed. The sequence is like this :

- First, go to forward direction for about a few seconds
- Than stop, while the chamber is still rotate
- Second, go back to reverse direction for about a few seconds
- Than stop, while the chamber is still rotate
- And so on, back and forth, until the the timer elapsed

5.14.1 SCHEMATIC

Timing sequence like the above description, can be implemented with many way, by using discrete electronic components, timer, using a program or a microcontroller or microprocessor, etc. Because I am learning the PIC microcontroller for right now, I will implement this function using this microcontroller, but for you who familiar with another kind of microcontroller my adapted it to your purpose. By using PIC micro, it can be made more compact. First I plan to make 2 buttons, 1 for set the timer and another for reset the timer or for the emergency stop push button. Then to know the timer works or not, I need a visual display. For this purpose I will use 7-segmen display showing the rest of the timer. To run the motor sequence of course I need a pair of relays (power relays, about 3 Amperes output), one for forward and another for reverse option. I will use the very common family of PIC micro, ie : 16F84A, because this is the most popular type and very simples used and very much used. Also can be obtained easily in the market. But this is the medium type of PIC micro family. It has 1kByte of memory (EEPROM type) and 13 I/O pins. It can be reprogrammable thousands times. Because the I/O just only 13 pins, I used a BCD to 7-segmen chip. So it will left a few I/O pins for expanded in the future. You can omitted this chip for timing sequence purpose and save one IC price, because the I/O just exactly enough.

- I/O port A-0 = SET push button
- I/O port A-1 = RST push button
- I/O port A-2 = Reserved
- I/O port A-3 = Reserved
- I/O port A-4 = Reserved
- I/O port B-0 = Forward Relay (Run motor forward)
- I/O port B-1 = Reverse Relay (Run motor reverse)
- I/O port B-2 = Activated unit 7-segmen (multiplexed)
- I/O port B-3 = Activated ten 7-segmen (multiplexed)
- I/O port B-4 = BCD data A (for 7-segmen)
- I/O port B-5 = BCD data B (for 7-segmen)
- I/O port B-6 = BCD data C (for 7-segmen)
- I/O port B-7 = BCD data D (for 7-segmen)

- Also integrated power supply to run it modularly

The I/O can be configured as input pin or output pin bit-ly. It is up to you to choose the I/O pin number goes to what function, but it infect the program firmware of course. Once you choose, then it is just like that, except you also change both, the program and the hardware.

5.14.2 Working of Washing Machine

The direction of rotation can be controlled When switchS1 is in position A, coil L1 of the motor receives the current directly, whereas coil L2 receives the current with a phase shiftdue to capacitor C. So the rotor rotates in clockwise direction (see Fig. 2(a)). Whenswitch S1 is in position B, the reverse happens and the rotor rotates in anti-clockwisedirection Thus switch
S1 can change the rotation direction.The motor cannot be reversed instantly. It needs abrief pause between switching directions, or else it mayget damaged. For this purpose, another spin direction control timer (IC2) is employed. It is realised with an IC 555. This timer gives an alternate ‘on’ and ‘off’ time duration of 10 seconds and 3 seconds, respectively.So after every l0 seconds of running (either in clockwise or anticlockwisedirection), the motor stops for a brief duration of 3 seconds. The values of R3 and R4 are calculated accordingly.The master timer is realised with monostable IC555 (IC1) and its ‘on’ time is decided by the resistance of 1-mega-ohm potmeterVR. A 47-kilo-ohm resistor is added in series so thateven when the VR knob is

in zero resistance position ,the net series resistanceis not zero.

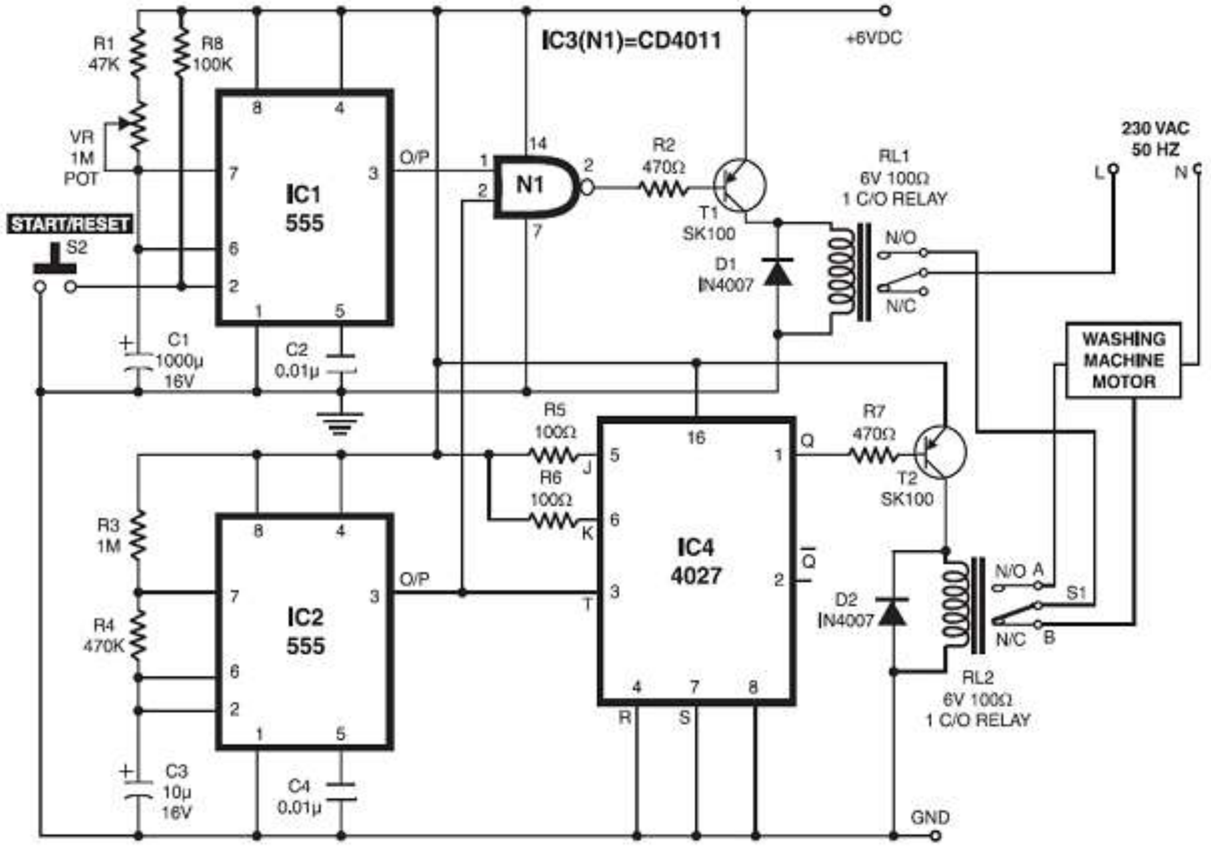


Fig 5.11 Circuit Diagram of Washing Machine

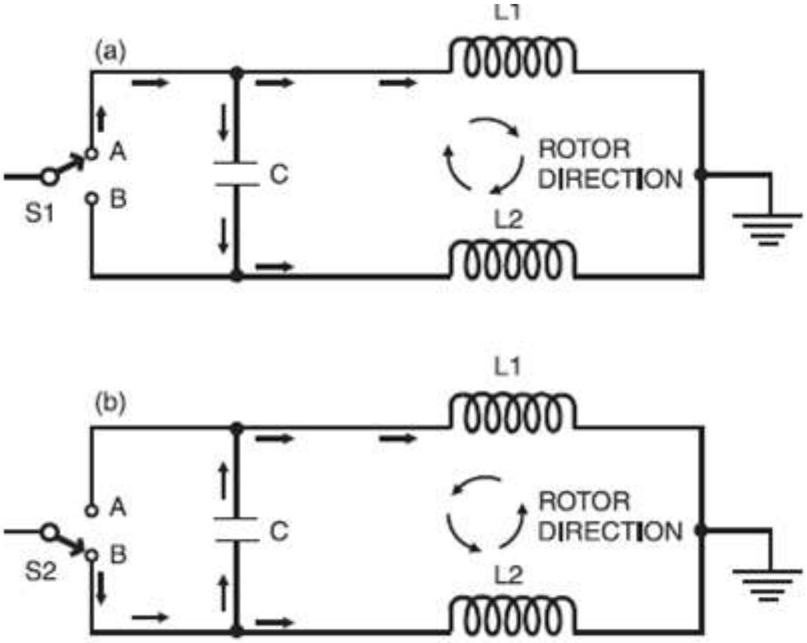


Fig 5.12 Rotation of Motor