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

SCHOOL OF ELECTRICAL AND ELECTRONICS
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

UNIT 1
Microprocessor , Interfacing and Its Applications-SECA1508

Basic Concepts of Microprocessors

Differences between: Microcomputer, Microprocessor and Microcontroller

- Microcomputer is a computer with a microprocessor as its CPU. Includes memory, I/O etc.
- Microprocessor is a silicon chip which includes ALU, register circuits & control circuits
- Microcontroller is a silicon chip which includes microprocessor, memory & I/O in a single package.

What is micro?

- Micro is a new addition.
  - In the late 1960's, processors were built using discrete elements.

    These devices performed the required operation, but were too large and too slow.

  - It went directly from discrete elements to a single chip. However, comparing today's microprocessors to the ones built in the early 1970's you find an extreme increase in the amount of integration.

What is a microprocessor?

- The word comes from the combination of micro and processor.
- Processor means a device that processes whatever. In this context processor means a device that processes numbers, specifically binary numbers, 0's and 1's.
- To process means to manipulate. It is a general term that describes all manipulation. Again in this content, it means to perform certain operations on the numbers that depend on the microprocessor's design. It is a programmable device that takes in numbers, performs on them arithmetic or logical operations according to the program stored in memory and then produces other numbers

As a Programmable device:

- The microprocessor can perform different sets of operations on the data it receives depending on the sequence of instructions supplied in the given program.
- By changing the program, the microprocessor manipulates the data in different ways as Instructions, Words, Bytes, etc.
- They processed information 8-bits at a time. That's why they are called ¯8-bit processors‖. They can handle large numbers, but in order to process these numbers, they broke them into 8-bit pieces and processed each group of 8-bits separately.

What is memory?

– Memory is the location where information is kept while not in current use. It is stored in memory

– Memory is a collection of storage devices. Usually, each storage device holds one bit. Also, in most kinds of memory, these storage devices are grouped into

groups of 8. These 8 storage locations can only be accessed together. So, one can only read or write in terms of bytes to and form memory.

– Memory is usually measured by the number of bytes it can hold. It is measured in Kilos, Megas and lately Gigas. A Kilo in computer language is 210 =1024. So, a KB (KiloByte) is 1024 bytes. Mega is 1024 Kilos and Giga is 1024 Mega.

– When a program is entered into a computer, it is stored in memory. Then as the microprocessor starts to execute the instructions, it brings the instructions from memory one at a time.

– Memory is also used to hold the data.

– The microprocessor reads (brings in) the data from memory when it needs it and writes (stores) the results into memory when it is done.

A MICROPROCESSOR-BASED SYSTEM

From the above description, we can draw the following block diagram to represent a microprocessor-based system as shown in fig 1

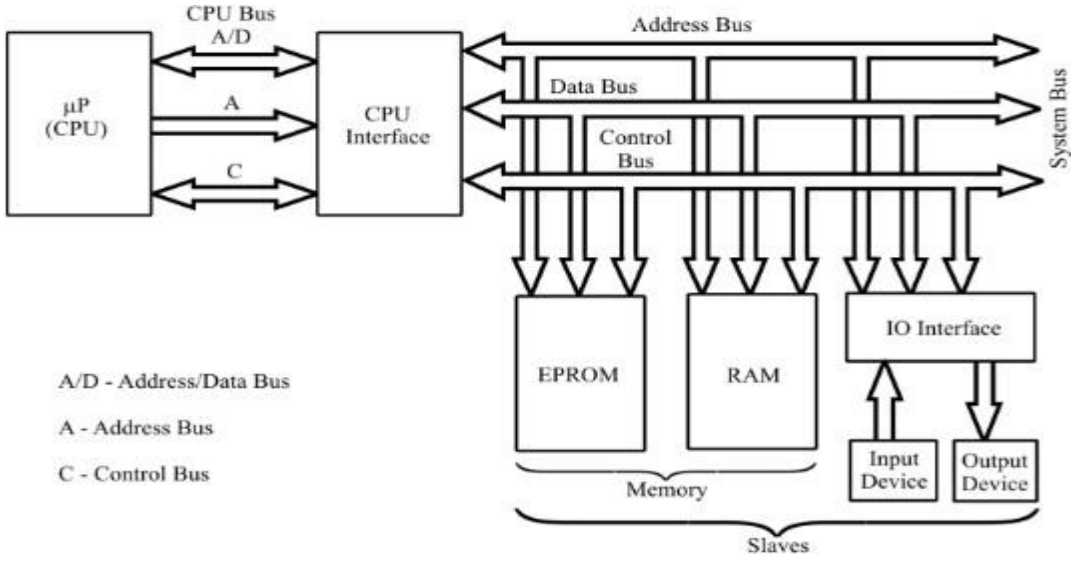


Fig.1.1 Microprocessor based system (organization of microcomputer)

In this system, the microprocessor is the master and all other peripherals are slaves. The master controls all peripherals and initiates all operations. The buses are group of lines that carry data, address or control signals. The CPU interface is provided to demultiplex the multiplexed lines, to generate the chip select signals and additional control signals. The system bus has separate lines for each signal.

All the slaves in the system are connected to the same system bus. At any time instant communication takes place between the master and one of the slaves. All the slaves have tristate logic and hence normally remain in high impedance state. The processor selects a slave by sending an address. When a slave is selected, it comes to the normal logic and communicates with the processor.

The EPROM memory is used to store permanent programs and data. The RAM memory is used to store temporary programs and data. The input device is used to enter program, data and to operate system. The output device is also used for examining the results. Since the speed of IO devices does not match with speed of microprocessor, an interface device is provided between system bus and IO device.

CENTRAL PROCESSING UNIT:

The CPU consists of ALU (Arithmetic and Logic Unit), Register unit and control unit. The CPU retrieves stored instructions and data word from memory; it also deposits processed data in memory.

a) ALU (Arithmetic and Logic Unit)

This section performs computing functions on data. These functions are arithmetic operations such as additions subtraction and logical operation such as AND, OR rotate etc. Result are stored either in registers or in memory or sent to output devices.

b) REGISTER UNIT:

It contains various register. The registers are used primarily to store data temporarily during the execution of a program. Some of the registers are accessible to the uses through instructions.

c) CONTROL UNIT:

It provides necessary timing & control signals necessary to all the operations in the microcomputer. It controls the flow of data between the p and peripherals (input, output & memory). The control unit gets a clock which determines the speed of the p.

The CPU has three basic functions

- It fetches an instructions word stored in memory.
- It determines what the instruction is telling it to do.(decodes the instruction)
- It executes the instruction. Executing the instruction may include some of the following major tasks.
    - Transfer of data from reg. to reg. in the CPU itself.
    - Transfer of data between a CPU reg. & specified memory location.
    - Performing arithmetic and logical operations on data from a specific memory location or a designated CPU register.
    - Directing the CPU to change a sequence of fetching instruction, if processing the data created a specific condition.
    - Performing housekeeping function within the CPU itself inorder to establish desired condition at certain registers.
    - It looks for control signal such as interrupts and provides appropriate responses.
    - It provides states, control, and timing signals that the memory and input/output section can use.

There are three buses:

Address Bus:

It is a group of wires or lines that are used to transfer the addresses of Memory or I/O devices. It is unidirectional. In Intel 8085 microprocessor, Address bus was of 16 bits. This means that Microprocessor 8085 can transfer maximum 16 bit address which means it can address 65,536 different memory locations. This bus is multiplexed with 8 bit data bus. So the most significant bits (MSB) of address goes through Address bus (A7-A0) and LSB goes through multiplexed data bus (AD0-AD7).

Data Bus:

Data Bus is used to transfer data within Microprocessor and Memory/Input or Output devices. It is bidirectional as Microprocessor requires to send or receive data. The data bus also works as address bus when multiplexed with lower order address bus. Data bus is 8 Bits long. The word length of a processor depends on data bus, thats why Intel 8085 is called 8 bit Microprocessor because it have an 8 bit data bus.

Control Bus:

Microprocessor uses control bus to process data that is what to do with the selected memory location. Some control signals are Read, Write and Opcode fetch etc. Various operations are performed by microprocessor with the help of control bus. This is a dedicated bus, because all timing signals are generated according to control signal. The microprocessor is the master, which controls all the activities of the system. To perform a specific job or task, the microprocessor has to execute a program stored in memory. The program consists of a set of instructions stored in consecutive memory location. In order to execute the program the microprocessor issues address and control signals, to fetch the instruction and data from memory one by one. After fetching each instruction it decodes the instruction and carries out the task specified by the instruction.

Memory

To execute a program:

- The user enters its instructions in binary format into the memory.
- The microprocessor then reads these instructions and whatever data is needed from memory, executes the instructions and places the results either in memory or produces it on an output device.

The three cycle instruction execution model

- To execute a program, the microprocessor ¯reads‖ each instruction from memory, ¯interprets‖ it, then ¯executes‖ it.
- To use the right names for the cycles
- The microprocessor fetches each instruction, decodes it, and then executes it. This sequence is continued until all instructions are performed.

The 8085 Machine Language

The 8085 (from Intel) is an 8-bit microprocessor. The 8085 uses a total of 246 bit patterns to form its instruction set. These 246 patterns represent only 74 instructions. The reason for the difference is that some (actually most) instructions have multiple different formats. Because it is very difficult to enter the bit patterns correctly, they are usually entered in hexadecimal instead of binary.

For example, the combination 0011 1100 which translates into ¯increment the number in the register called the accumulator‖, is usually entered as 3C.

Assembly Language

Entering the instructions using hexadecimal is quite easier than entering the binary combinations. However, it still is difficult to understand what a program written in hexadecimal does. So, each company defines a symbolic code for the instructions.

These codes are called ¯mnemonics‖.

The mnemonic for each instruction is usually a group of letters that suggest the operation performed.

- Using the same example from before,

  00111100 translates to 3C in hexadecimal (OPCODE)

  Its mnemonic is: ¯INR A‖.

  INR stands for ¯increment register‖ and A is short for accumulator.

It is important to remember that a machine language and its associated assembly language are completely machine dependent. In other words, they are not transferable from one microprocessor to a different one.

Assembling" The Program

How does assembly language get translated into machine language?

– There are two ways:

– 1st there is ¯hand assembly‖.

- The programmer translates each assembly language instruction into its equivalent hexadecimal code (machine language). Then the hexadecimal code is entered into memory.
- The other possibility is a program called an ¯assembler‖, which does the translation automatically.

8085 MICROPROCESSOR ARCHITECTURE

Features of 8085

- 8-bit general purpose µp

- Capable of addressing 64 k of memory
- Has 40 pins as shown in fig 2
- Requires +5 v power supply
- Can operate with 3 MHz clock
- 8085 upward compatible

Pin Diagram of 8085

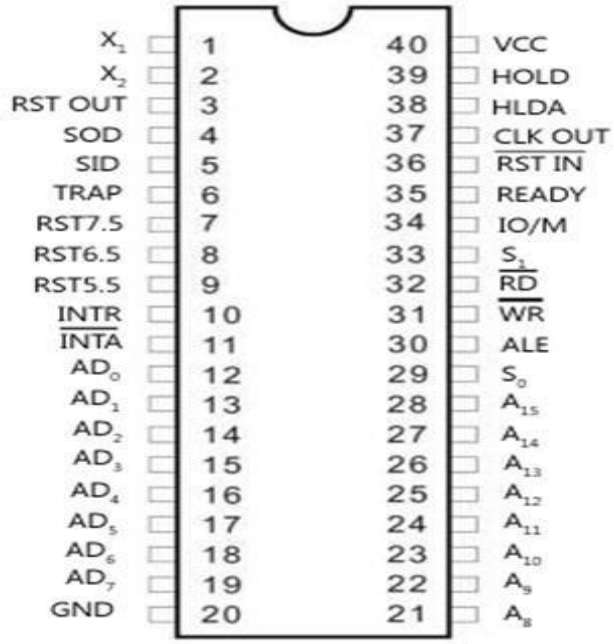


Fig 1.2 Pin Diagram of 8085

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 the

data 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. 3 stated 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 for Ready to go high before completing the read or write cycle.

HOLD (Input)

It indicates that another Master is requesting the use of the Address and Data Buses. 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 processor can regain the buses only after the Hold is removed. When the Hold is acknowledged, the Address, Data, RD, WR, and IO/M lines are stated.

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 only during 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) RD during 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 as INTR. 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 CPU is being reset also 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 to give 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 I/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.

Signal Classification of 8085

The signal Classification of 8085 is as shown in fig3.

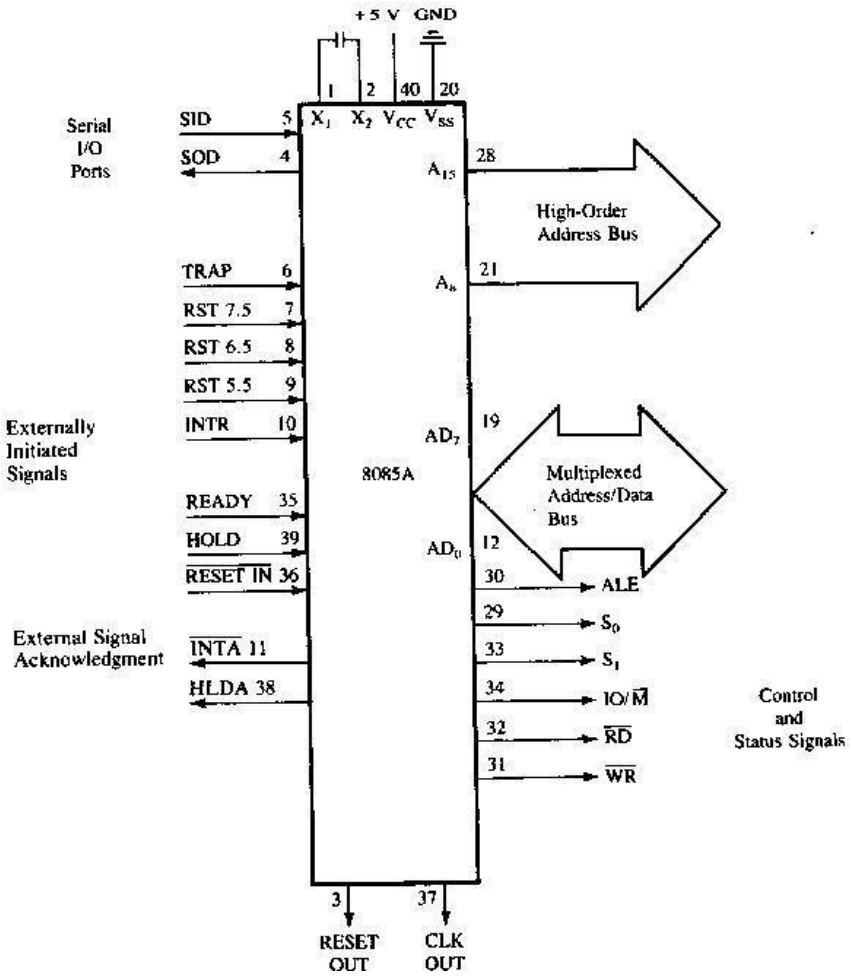


Fig:1. 3 Signal Classifications of 8085

System Bus – wires connecting memory & I/O to microprocessor

– ADDRESS BUS

- Unidirectional
- Identifying peripheral or memory location

– DATA BUS

- Bidirectional
- Transferring data

– CONTROL BUS

- Synchronization signals
- Timing signals
- Control signal

ARCHITECTURE OF INTEL 8085 MICROPROCESSOR

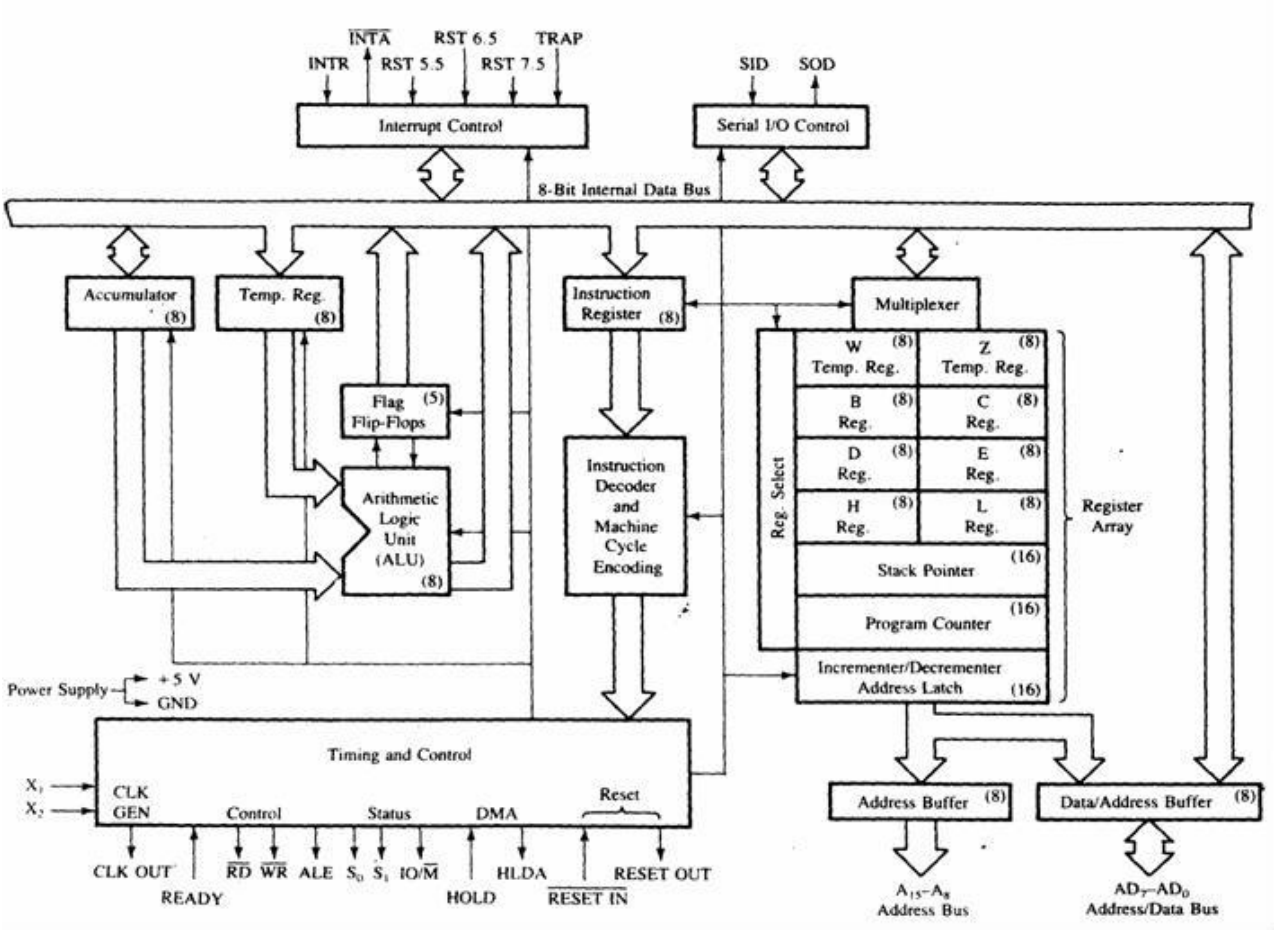


Fig:1.4 Architecture of intel 8085 microprocessor

The architecture of INTEL 8085 microprocessor is as shown in fig4.

Intel 8085 Microprocessor

Microprocessor consists of:

– Control unit: control microprocessor operations.

– ALU: performs data processing function.

– Registers: provide storage internal to CPU.

– Interrupts

– Internal data bus

The ALU

• In addition to the arithmetic & logic circuits, the ALU includes the accumulator, which is part of every arithmetic & logic operation.

• Also, the ALU includes a temporary register used for holding data temporarily during the execution of the operation. This temporary register is not accessible by the programmer.

Registers

General Purpose Registers

- B, C, D, E, H & L (8 bit registers)
- Can be used singly
- Or can be used as 16 bit register pairs

BC, DE, HL

- H & L can be used as a data pointer (holds memory address)
- Special Purpose Registers
- Accumulator (8 bit register)

- Store 8 bit data
- Store the result of an operation
- Store 8 bit data during I/O transfer Address

Flag Register

– 8 bit register – shows the status of the microprocessor before/after an operation

– S (sign flag), Z (zero flag), AC (auxillary carry flag), P (parity flag) & CY (carry flag)

D7	D6	D5	D4	D3	D2	D1	D0
S	Z	X	AC	X	P	X	CY

Sign Flag

- Used for indicating the sign of the data in the accumulator
- The sign flag is set if negative (1 – negative)
- The sign flag is reset if positive (0 –positive)

Zero Flag

- Is set if result obtained after an operation is 0
- Is set following an increment or decrement operation of that register

Carry Flag

```
    10110011
+   01001101
    --------
    00000000
```

- Is set if there is a carry or borrow from arithmetic operation

	1011 0101		1011 0101
+	0110 1100	-	1100 1100
	---------		---------
Carry 1	0010 0001	Borrow 1	1110 1001

Auxillary Carry Flag

– Is set if there is a carry out of bit 3

Parity Flag

- Is set if parity is even
- Is cleared if parity is odd

The Internal Architecture

We have already discussed the general purpose registers, the Accumulator, and the flags.

The Program Counter (PC)

– This is a register that is used to control the sequencing of the execution of instructions.

– This register always holds the address of the next instruction.

– Since it holds an address, it must be 16 bits wide.

The Stack pointer

– The stack pointer is also a 16-bit register that is used to point into memory.

– The memory this register points to is a special area called the stack.

– The stack is an area of memory used to hold data that will be retreived soon.

– The stack is usually accessed in a Last In First Out (LIFO) fashion.

Non Programmable Registers

Instruction Register & Decoder

– Instruction is stored in IR after fetched by processor

– Decoder decodes instruction in IR

Internal Clock generator

– 3.125 MHz internally

– 6.25 MHz externally

The Address and Data Busses

•The address bus has 8 signal lines A8 – A15 which are unidirectional.

•The other 8 address bits are multiplexed (time shared) with the 8 data bits.

So, the bits AD0 – AD7 are bi-directional and serve as A0 – A7 and D0 – D7 at the same time.

•During the execution of the instruction, these lines carry the address bits during the early part, then during the late parts of the execution, they carry the 8 data bits.

In order to separate the address from the data, we can use a latch to save the value before the function of the bits changes.

Demultiplexing AD7-AD0

– From the above description, it becomes obvious that the AD7- AD0 lines are serving a dual purpose and that they need to be demultiplexed to get all the information.

– The high order bits of the address remain on the bus for three clock periods. However, the low order bits remain for only one clock period and they would be lost if they are not saved externally. Also, notice that the low order bits of the address disappear when they are needed most.

– To make sure we have the entire address for the full three clock cycles, we will use an external latch to save the value of AD7– AD0 when it is carrying the address bits. We use the ALE signal to enable this latch.

Demultiplexing AD7-AD0

Given that ALE operates as a pulse during T1, we will be able to latch the address. Then when ALE goes low, the address is saved and the AD7– AD0 lines can be used for their purpose as the bi-directional data lines.

Demultiplexing the Bus AD7 – AD0

- The high order address is placed on the address bus and hold for 3 clk periods,
- The low order address is lost after the first clk period, this address needs to be hold however we need to use latch
- The address AD7 – AD0 is connected as inputs to the latch 74LS373.
- The ALE signal is connected to the enable (G) pin of the latch and the OC – Output control – of the latch is grounded

ADDRESSING MODES

The microprocessor has different ways of specifying the data for the instruction. These are called ¯addressing modes‖.

The 8085 has four addressing modes:

– Implied CMA

– Immediate MVI B, 45

– Direct LDA 4000

– Indirect LDAX B

Load the accumulator with the contents of the memory location whose address is stored in the register pair BC).

Many instructions require two operands for execution. For example transfer of data between two registers. The method of identifying the operands position by the instruction format is known as the addressing mode. When two operands are involved in an instruction, the first operand is assumed to be in a register Mp itself.

Types of Addressing Modes

- Register addressing
- Direct addressing mode
- Register indirect addressing
- Immediate Addressing mode
- Implied addressing mode

Register Addressing

This type of addressing mode specifies register or register pair that contains data.ie (only the register need be specified as the address of the operands).

Example MOV B, A (the content of A is copied into the register B)

Direct Addressing Mode

Data is directly copied from the given address to the register.

Example LDA 3000H (The content at the location 3000H is copied to the register A).

Register Indirect Addressing

In this mode, the address of operand is specified by a register pair

Example MOV A, M (Move data from memory location specified by H-L pair to accumulator)

Immediate Addressing Mode

In this mode, the operand is specified within the instruction itself.

Example MVI A, 05 H (Move 05 H in accumulator.)

Implied Addressing Mode

This mode doesn't require any operand. The data is specified by opcode itself.

Example RAL,

CMP

INSTRUCTION SET OF 8085

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called Instruction Set. Since the 8085 is an 8-bit device it can have up to 28 (256) instructions. However, the 8085 only uses 246 combinations that represent a total of 74 instructions. Each instruction has two parts. The first part is the task or operation to be performed. This part is called the ¯opcode‖ (operation code). The second part is the data to be operated on. This part is called the ¯operand‖.

Instruction Size

• Depending on the operand type, the instruction may have different sizes. It will occupy a different number of memory bytes.

– Typically, all instructions occupy one byte only.

– The exception is any instruction that contains immediate data or a memory address.

• Instructions that include immediate data use two bytes.

– One for the opcode and the other for the 8-bit data.

- Instructions that include a memory address occupy three bytes.

– One for the opcode, and the other two for the 16-bit address.

Classification of Instruction Set

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

DATA TRANSFER INSTRUCTIONS

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

This instruction copies the contents of the source register into the destination register. The contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. Example: MOV B, C or MOV B, M.

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

The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the H-L registers. Example: MVI B, 57H or MVI M, 57H.

Opcode	Operand	Description
LDA	16-bit address	Load Accumulator

The contents of a memory location, specified by a 16- bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034H

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

The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. Example: LDAX B

Opcode	Operand	Description
STA	16-bit address	Store accumulator direct

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

Opcode	Operand	Description
STAX	Reg. pair	Store accumulator indirect

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

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

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

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

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

Opcode	Operand	Description
SPHL	None	Copy H-L pair to the Stack Pointer (SP)

This instruction loads the contents of H-L pair into SP. Example: SPHL

Opcode	Operand	Description
XTHL	None	Exchange H-L with top of stack

The contents of L register are exchanged with the location pointed out by the contents of the SP. The contents of H register are exchanged with the next location (SP + 1). Example: XTHL

Opcode	Operand	Description
PCHL	None	Load program counter with H-L contents

The contents of registers H and L are copied into the program counter (PC). The contents of H are placed as the high-order byte and the contents of L as the low-order byte. Example: PCHL

Opcode	Operand	Description
PUSH	Reg. pair	Push register pair onto stack

The contents of register pair are copied onto stack. SP is decremented and the contents of high-order registers (B, D, H, A) are copied into stack. SP is again decremented and the contents of low-order registers (C, E, L, Flags) are copied into stack. Example: PUSH B

Opcode	Operand	Description
POP	Reg. pair	Pop stack to register pair

The contents of top of stack are copied into register pair. The contents of location pointed out by SP are copied to the low-order register (C, E, L, Flags). SP is incremented and the contents of location are copied to the high-order register (B, D, H, A). Example: POP H

Opcode	Operand	Description
OUT	8-bit port address	Copy data from accumulator to a port with 8- bit address

The contents of accumulator are copied into the I/O port. Example: OUT 78 H

Opcode	Operand	Description
IN	8-bit port address	Copy data to accumulator from a port with 8- bit address

The contents of I/O port are copied into accumulator. Example: IN 8C H

1. ARITHMETIC INSTRUCTIONS

These instructions perform arithmetic operations such as addition, subtraction, increment, and decrement.

Addition

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

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

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

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

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

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

The 8-bit data is added to the contents of accumulator. The result is stored in accumulator. All flags are modified to reflect the result of the addition. Example: ADI 45 H

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

The 8-bit data and the Carry Flag (CY) are added to the contents of accumulator. The result is stored in accumulator. All flags are modified to reflect the result of the addition. Example: ACI 45 H

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

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

Subtraction

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

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

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

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

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

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

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

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

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

Increment/Decrement

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

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

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

Opcode	Operand	Description
INX	R	Increment register pair by 1

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

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

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

Opcode	Operand	Description
DCX	R	Decrement register pair by 1

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

2. LOGICAL INSTRUCTIONS

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

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

AND, OR, XOR

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

- AND operation
- OR operation
- XOR operation

with the contents of accumulator. The result is stored in accumulator.

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

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

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

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

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

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

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

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

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

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

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

The contents of the accumulator are XORedwith the 8-bit data. The result is placed in the accumulator. S, Z, P are modified to reflect the result. CY and AC are reset. Example: XRI 86H.

Rotate

Each bit in the accumulator can be shifted either left or right to the next position as shown in fig5.

Opcode	Operand	Description
RLC	None	Rotate accumulator left

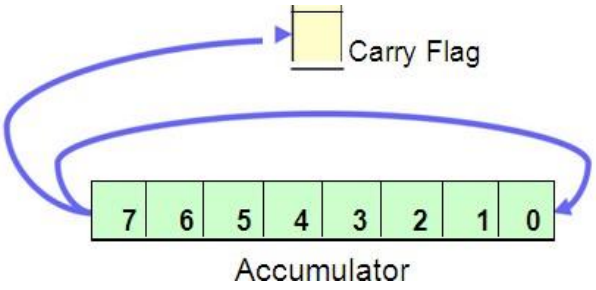


Fig 1.5. : Work flow of RLC

Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7. S, Z, P, AC are not affected. Example: RLC.

Opcode	Operand	Description
RRC	None	Rotate accumulator right

Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0. S, Z, P, AC are not affected. Example: RRC.

Opcode	Operand	Description
RAL	None	Rotate accumulator left through carry

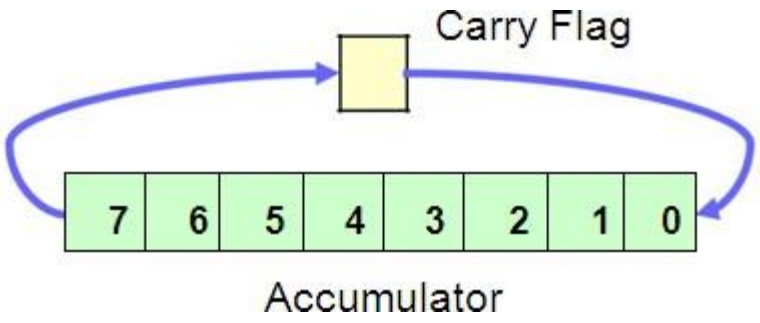


Fig 1.6. : Work flow of RAL

Each binary bit of the accumulator is rotated left by one position through the Carry flag as shown in fig 6. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7. S, Z, P, AC are not affected. Example: RAL.

Opcode	Operand	Description
RAR	None	Rotate accumulator right through carry

Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0. S, Z, P, AC are not affected. Example: RAR.

COMPARE

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

- Equality
- Greater Than
- Less Than

with the contents of accumulator. The result is reflected in status flags.

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

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

- if (A) < (reg/mem): carry flag is set
- if (A) = (reg/mem): zero flag is set
- if (A) > (reg/mem): carry and zero flags are reset.

Example: CMP B or CMP M

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

The 8-bit data is compared with the contents of accumulator.The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:

- if (A) < data: carry flag is set
- if (A) = data: zero flag is set
- if (A) > data: carry and zero flags are reset

Example: CPI 89H

COMPLEMENT

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

Opcode	Operand	Description
CMA	None	Complement accumulator

The contents of the accumulator are complemented. No flags are affected. Example: CMA.

Opcode	Operand	Description
CMC	None	Complement carry

The Carry flag is complemented. No other flags are affected. Example: CMC.

Opcode	Operand	Description
STC	None	Set carry

The Carry flag is set to 1. No other flags are affected. Example: STC.

3. BRANCHING INSTRUCTIONS

The branching instruction alters the normal sequential flow. These instructions alter either unconditionally or conditionally.

Branch operations are of two types:

Unconditional branch-- Go to a new location no matter what.

Conditional branch-- Go to a new location if the condition is true.

Opcode	Operand	Description
JMP	16-bit address	Jump unconditionally

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.

Example: JMP 2034 H.

Opcode	Operand	Description
Jx	16-bit address	Jump conditionally

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW. Replace x with condition

Example: JZ 2034 H.

Jump conditionally

Opcode	**Description**	**Status flag**
JC	Jump if Carry	CY = 1
JNC	Jump if no carry	CY=0
JP	Jump if positive	S=0
JM	Jump if minus	S=1
JZ	Jump if Zero	Z=1
JNZ	Jump if no zero	Z=0
JPE	Jump if parity even	P=1
JPO	Jump if parity odd	P=0

Opcode	Operand	Description
CALL	16-bit address	Call unconditionally

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

Example: CALL 2034 H.

Opcode	Operand	Description
Cx	16-bit address	Call conditionally

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW. Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack. Replace x with condition

Example: CZ 2034 H.

Call Conditionally

Opcode	**Description**	**Status flag**
CC	Call if carry	CY=1
CNC	Call if no carry	CY=0
CP	Call if positive	S=0
CM	Call if minus	S=1
CZ	Call if Zero	Z=1
CNZ	Call if no zero	Z=0
CPE	Call if parity even	P=1
CPO	Call if parity odd	P=0

Opcode	Operand	Description
RET	None	Return unconditionally

The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.

Example: RET.

Opcode	Operand	Description
Rx	None	Call conditionally

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

RETURN CONDITIONALLY

Opcode	**Description**	**Status flag**
RC	Return if carry	CY=1
RNC	Return if no carry	CY=0
RP	Return Call if positive	S=0
RM	Return if minus	S=1
RZ	Return if Zero	Z=1
RNZ	Return if no zero	Z=0
RPE	Return if parity even	P=1
RPO	Return if parity odd	P=0

Opcode	Operand	Description
RST	0-7	Restart (Software Interrupts)

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

RESTART Address table

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

4. MACHINE CONTROL INSTRUCTIONS

The control instructions control the operation of microprocessor.

Opcode	Operand	Description
NOP	None	No operation

No operation is performed. The instruction is fetched and decoded but no operation is executed. Usually used for delay or to replace instructions during debugging.

Example: NOP

Opcode	Operand	Description
HLT	None	Halt

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

Opcode	Operand	Description
DI	None	Disable interrupt

The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled.

No flags are affected.

Example: DI

Opcode	Operand	Description
EI	None	Enable interrupt

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

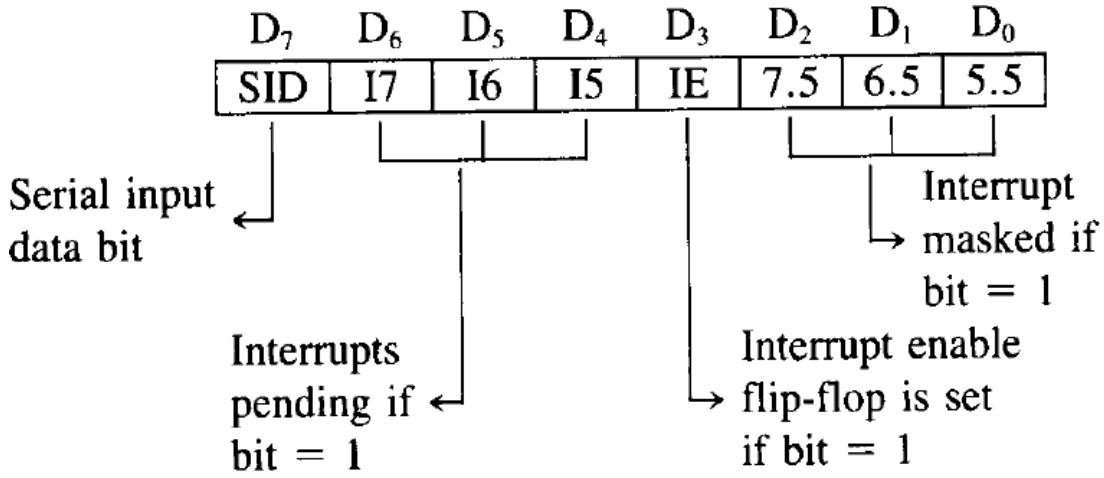
Example: EI

Opcode	Operand	Description
RIM	None	Read Interrupt Mask

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

Example: RIM

RIM Instruction

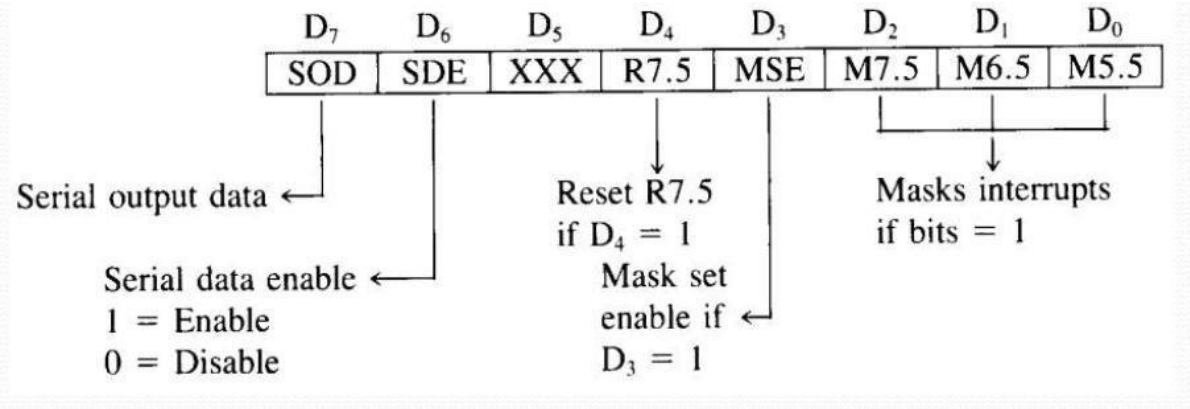


Opcode	Operand	Description
SIM	None	Set Interrupt Mask

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

Example: SIM

SIM Instruction



TIMING DIAGRAM

Timing diagram is the display of initiation of read/write and transfer of data operations under the control of 3-status signals IO / M, S1, and S0. All actions in the microprocessor are controlled by either leading or trailing edge of the clock.

,

Machine Cycle

It is the time required by the microprocessor to complete the operation of accessing the memory devices or I/O devices. In machine cycle various operations like opcode fetch, memory read, memory write, I/O read, I/O write are performed.

T-State

Each clock cycle is called as T-states.

Each machine cycle is composed of many clock cycles. Since, the data and instructions, both are stored in the memory, the μP performs fetch operation to read the instruction or data and then execute the instruction. The 3-status signals: IO / M, S1, and S0 are generated at the beginning of each machine cycle. The unique combination of these 3-status signals identify read or write operation and remain valid for the duration of the cycle.

Table 1 Machine Cycle Status And Control Signals

	Status			*Controls*		
Machine cycle	$IO / \overline{M}$	S_1	S_0	$\overline{RD}$	$\overline{WR}$	$\overline{INTA}$
Opcode Fetch (OF)	0	1	1	0	1	1
Memory Read	0	1	0	0	1	1
Memory Write	0	0	1	1	0	1
I/O Read (I/OR)	1	1	0	0	1	1
I/O Write (I/OW)	1	0	1	1	0	1
Acknowledge of INTR (INTA)	1	1	1	1	1	0
BUS Idle (BI) : DAD	0	1	0	1	1	1
ACK of RST, TRAP	1	1	1	1	1	1
HALT	Z	0	0	Z	Z	1
HOLD	Z	X	X	Z	Z	1

X ⇒ Unspecified, and Z ⇒ High impedance state

Table1 shows details of the unique combination of these status signals to identify different machine cycles. Thus, time taken by any μP to execute one instruction is calculated in terms of the clock period. The execution of instruction always requires read and writes operations to transfer data to or from the μP and memory or I/O devices. Each read/ write operation constitutes one machine cycle (MC1) as indicated in Fig.7. Each machine cycle consists of many clock periods/ cycles, called T-states.

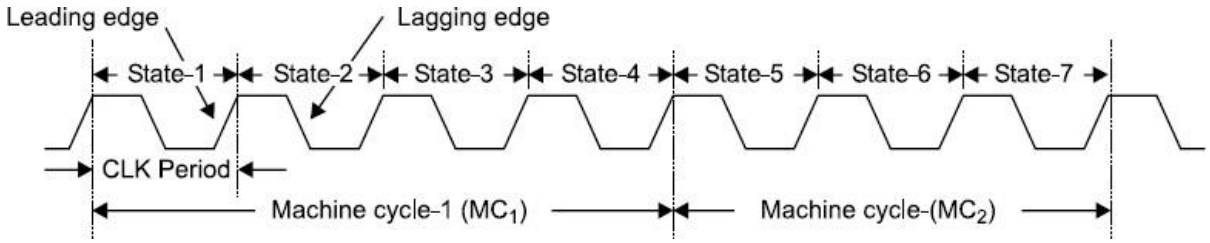


Fig1.7 Machine cycle showing clock periods

Processor Cycle:

The functions of the microprocessor are divided into fetch and execute cycle of any instruction of a program. The program is nothing but number of instructions stored in the memory in sequence. In the normal process of operation, the microprocessor fetches (receives or reads) and executes one instruction at a time in the sequence until it executes the halt (HLT) instruction.

Instruction Cycle

An instruction cycle is defined as the time required to fetch and execute an instruction. For executing any program, basically 2-steps are followed sequentially with the help of clocks

- Fetch, and
- Execute.

The time taken by the μP in performing the fetch and execute operations are called fetch and execute cycle. Thus, sum of the fetch and execute cycle is called the

instruction cycle as indicated in Fig. 8. Each read or writes operation constitutes a machine cycle. The instructions of 8085 require 1-5 machine cycles containing 3-6 states (clocks). The 1st machine cycle of any instruction is always an Op Code fetch cycle in which the processor decides the nature of instruction. It is of at least 4-states. It may go up to 6-states.

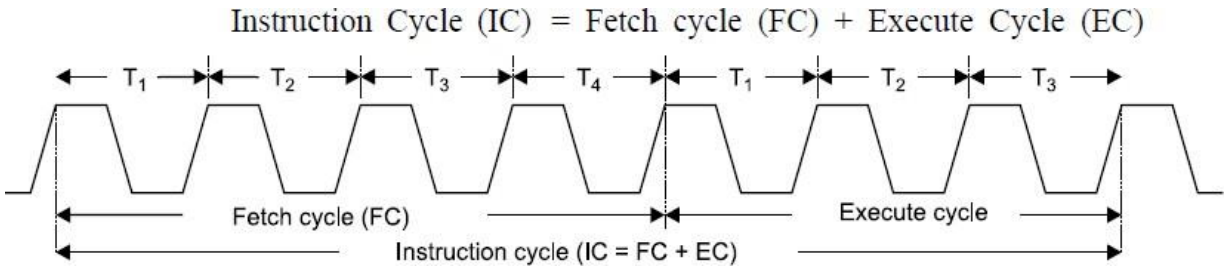


Fig1.8 Processor cycle

Rules To Identify Number Of Machine Cycles In An Instruction:

- If an addressing mode is direct, immediate or implicit then No. of machine cycles = No. of bytes.
- If the addressing mode is indirect then No. of machine cycles = No. of bytes + 1. Add +1 to the No. of machine cycles if it is memory read/write operation.
- If the operand is 8-bit or 16-bit address then, No. of machine cycles = No. of bytes +1.
- These rules are applicable to 80% of the instructions of 8085.

Timing Diagram of Opcode Fetch

The process of Opcode fetch operation requires minimum 4-clock cycles T1, T2, T3, and T4 and is the 1st machine cycle (M1) of every instruction.

Example

Fetch a byte 41H stored at memory location 2105H.

For fetching a byte, the microprocessor must find out the memory location where it is stored. Then provide condition (control) for data flow from memory to the microprocessor. The process of data flow and timing diagram of fetch operation are shown in Fig. 9. The microprocessor fetches Opcode of the instruction from the memory as per the sequence below

- A low IO/M means microprocessor wants to communicate with memory.
- The microprocessor sends a high on status signal S1 and S0 indicating fetch operation.
- The microprocessor sends 16-bit address. AD bus has address in 1st clock of the 1st machine cycle, T1.
- AD7 to AD0 address is latched in the external latch when ALE = 1.
- AD bus now can carry data.
- In T2, the RD control signal becomes low to enable the memory for read operation.
- The memory places opcode on the AD bus
- The data is placed in the data register (DR) and then it is transferred to IR.
- During T3 the RD signal becomes high and memory is disabled.
- During T4 the opcode is sent for decoding and decoded in T4.
- The execution is also completed in T4 if the instruction is single byte.
- More machine cycles are essential for 2- or 3-byte instructions. The 1st machine cycle M1 is meant for fetching the opcode. The machine cycles M2 and M3 are required either read/ write data or address from the memory or I/O devices.

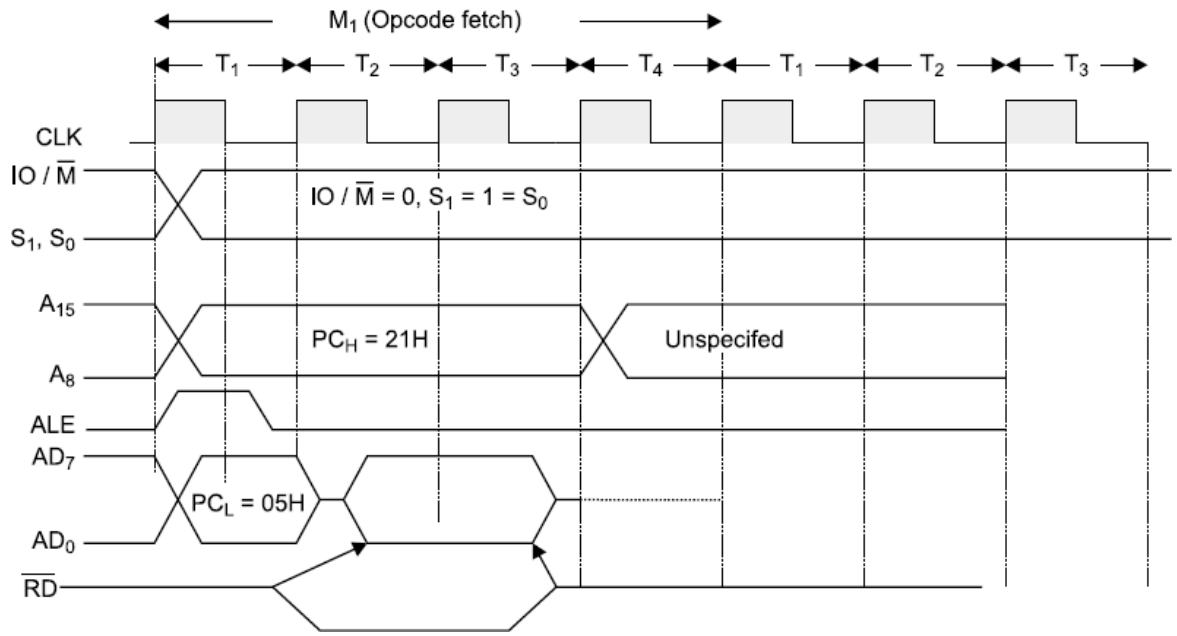


Fig:1.9 Opcode fetch

Example For Opcode Fetch

- Explain the execution of MVI B, 05H stored at locations indicated below
-

Mnemonics	Machine code	Memory Locations
MVI B, 05H	06H	1000H
	05H	1001H

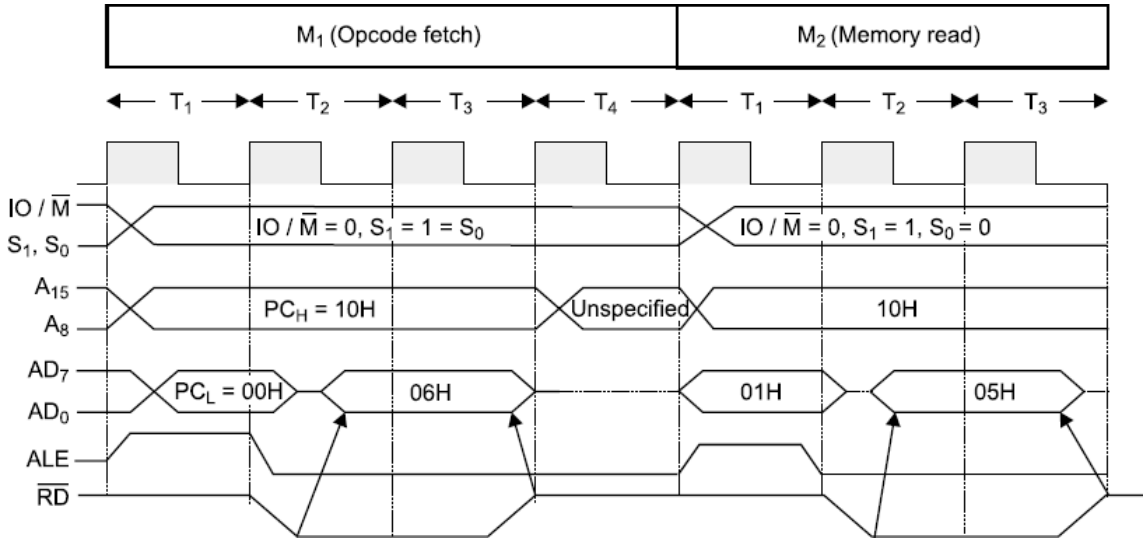


Fig. 1.10 Timing diagram for MVI B,05H

The MVI B, 05H instruction requires 2-machine cycles (M1 and M2). M1 requires 4-states and M2 requires 3-states, total of 7-states as shown in Fig. 10. Status signals IO/M, S1 and S0 specifies the 1st machine cycle as the op-code fetch. In T1-state, the high order address {10H} is placed on the bus A15 ⇔A8 and low-order address {00H} on the bus AD7 ⇔AD0 and ALE = 1. In T2 -state, the RD line goes low and the data 06 H from memory location 1000H are placed on the data bus. The fetch cycle becomes complete in T3-state. The instruction is decoded in the T4-state. During T4-state, the contents of the bus are unknown. With the change in the status signal, IO/M = 0, S1 = 1 and S0 = 0, the 2nd machine cycle is identified as the memory read. The address is 1001H and the data byte [05H] is fetched via the data bus. Both M1 and M2 perform memory read operation, but the M1 is called op-code fetch i.e., the 1st machine cycle of each instruction is identified as the opcode fetch cycle.

Mnemonic	*Instruction Byte*	*Machine Cycle*	*T-sstates*
MVI B,05H	Opcode	Opcode Fetch	4
	Immediate Data	Read Immediate Data	3
			7

TIMING DIAGRAM OF MEMORY READ

Operation:

- It is used to fetch one byte from the memory.
- It requires 3 T-States.
- It can be used to fetch operand or data from the memory.
- During T1, A8-A15 contains higher byte of address. At the same time ALE is high. Therefore Lower byte of address A0-A7 is selected from AD0-AD7 as shown in fig 11.
- Since it is memory ready operation, IO/M (bar) goes low.
- During T2 ALE goes low, RD (bar) goes low. Address is removed from AD0-AD7 and data D0-D7 appears on AD0-AD7.
- During T3, Data remains on AD0-AD7 till RD (bar) is at low signal.

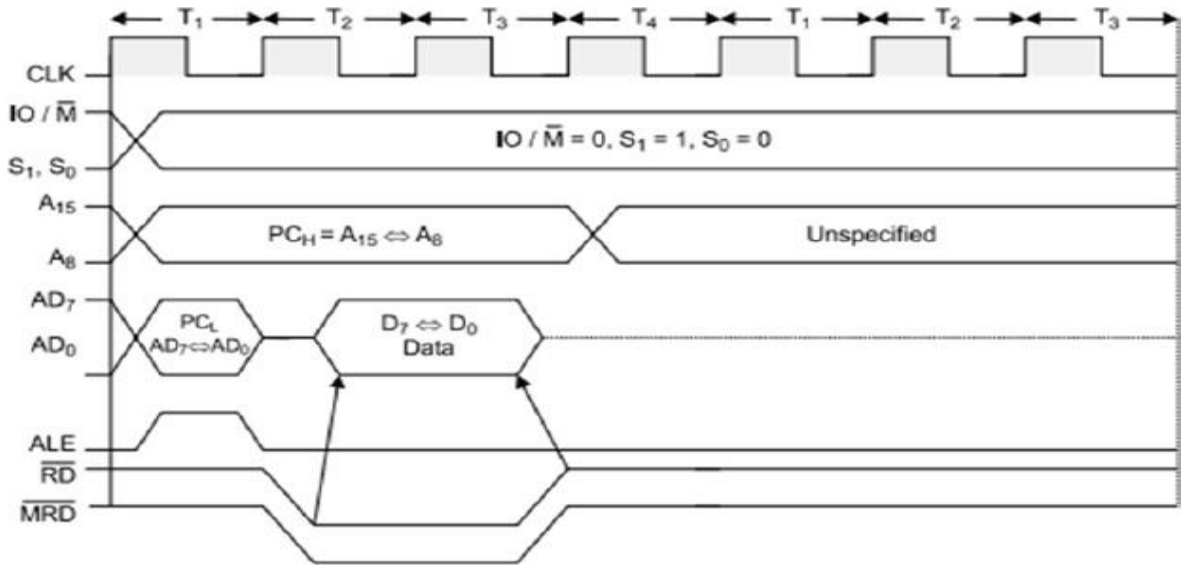


Fig:1.11.Memory read timing

diagramTIMING DIAGRAM OF MEMORY WRITE

Operation:

- It is used to send one byte into memory.
- It requires 3 T-States.
- During T1, ALE is high and contains lower address A0-A7 from AD0-AD7.
- A8-A15 contains higher byte of address.
- As it is memory operation, IO/M (bar) goes low.
- During T2, ALE goes low, WR (bar) goes low and Address is removed from AD0-AD7 and then data appears on AD0-AD7 as in fig 12.
- Data remains on AD0-AD7 till WR (bar) is low.

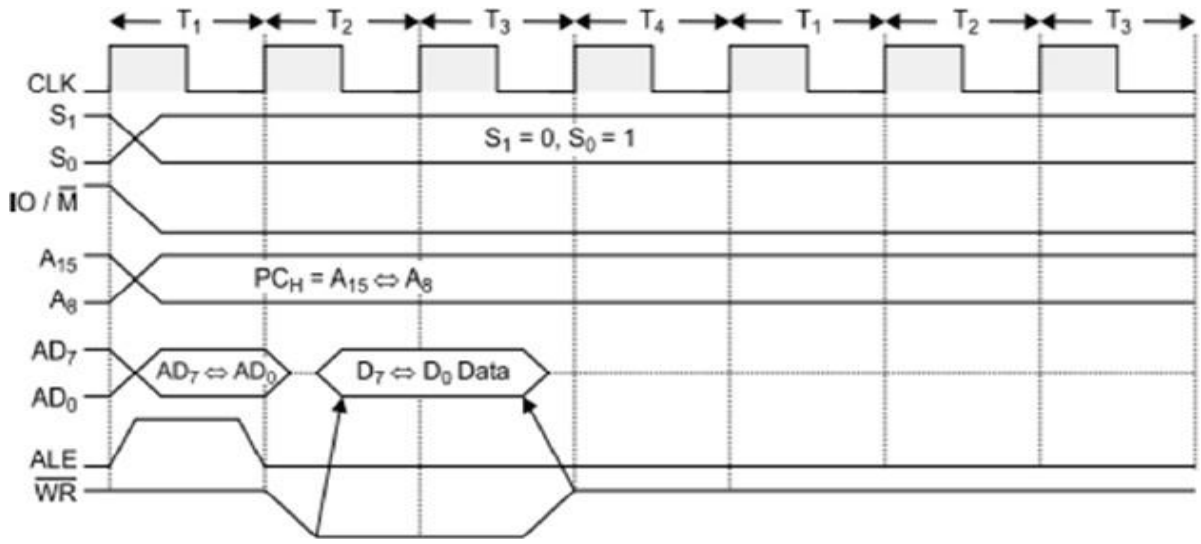


Fig 1.12.Memory Write timing

diagramTIMING DIAGRAM OF IO READ

Operation:

- It is used to fetch one byte from an IO port.
- It requires 3 T-States.
- During T1, The Lower Byte of IO address is duplicated into higher order address bus A8-A15 as in fig13.
- ALE is high and AD0-AD7 contains address of IO device.
- IO/M (bar) goes high as it is an IO operation.
- During T2, ALE goes low, RD (bar) goes low and data appears on AD0-AD7 as input from IO device.
- During T3 Data remains on AD0-AD7 till RD (bar) is low.

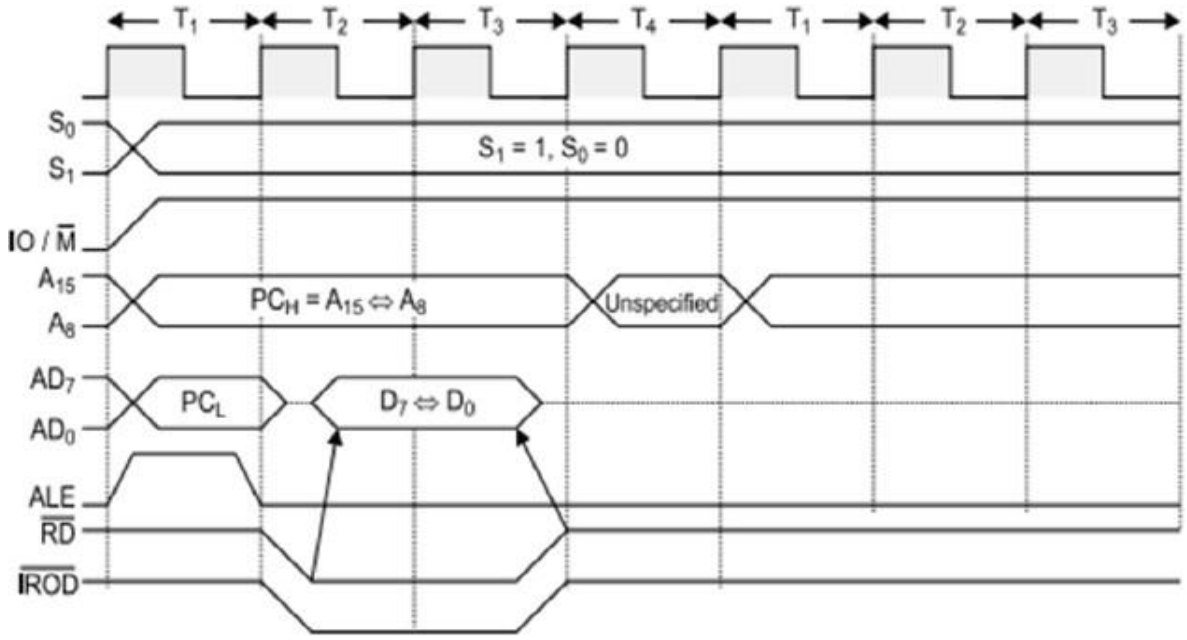


Fig 1.13.IO Read timing

diagramTIMING DIAGRAM OF IO WRITE

Operation:

- It is used to writ one byte into IO device.
- It requires 3 T-States.
- During T1, the lower byte of address is duplicated into higher order address bus A8-A15 as in fig 14.
- ALE is high and A0-A7 address is selected from AD0-AD7.
- As it is an IO operation IO/M (bar) goes low.
- During T2, ALE goes low, WR (bar) goes low and data appears on AD0-AD7 to write data into IO device.
- During T3, Data remains on AD0-AD7 till WR(bar) is low.

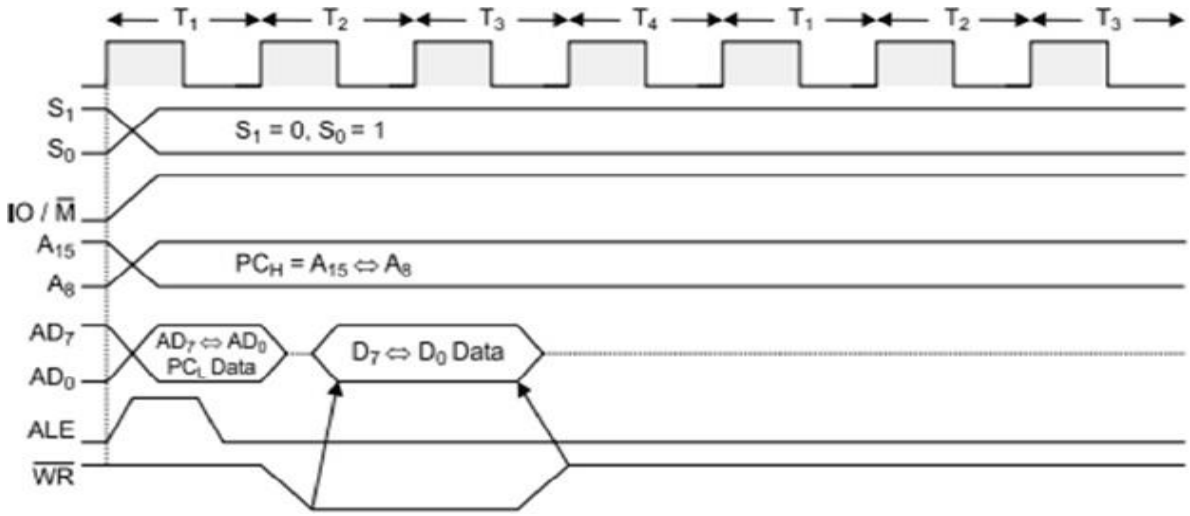


Fig 1.14. IO Write timing diagram

INTERRUPT:

An interrupt is a signal initiated by an external device to the microprocessor. Once this signal is received, the microprocessor completes the execution of the current instruction and responds to the interrupt

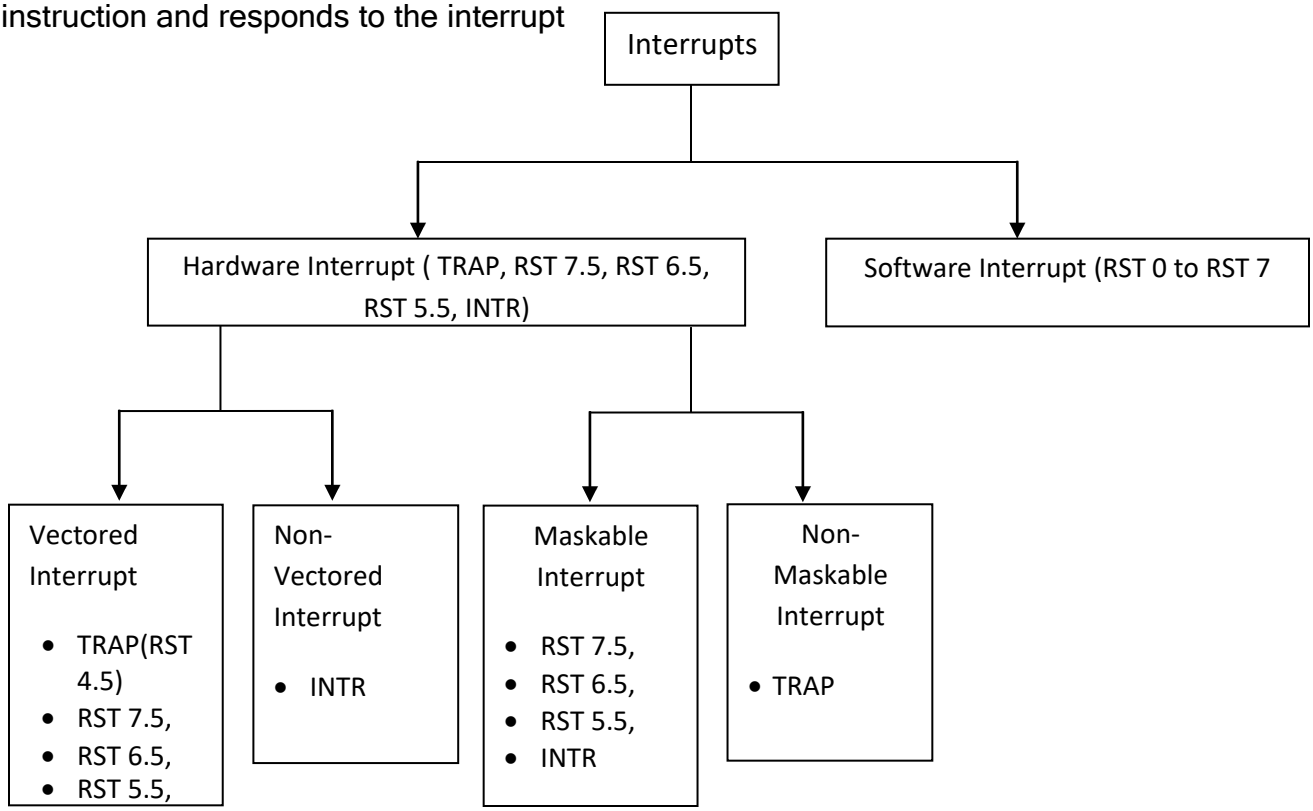


Fig 1.15. Structured classification of interrupts

SOFTWARE INTERRUPTS OF 8085

The software interrupts are program instructions. When the instruction is executed, the processor executes an interrupt service routine stored in the vector address of the software interrupt instruction. The software interrupts of 8085 are RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6 and RST 7.

The vector addresses of software interrupts are given in table below

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

Interrupt	Vector address
RST 7.5	$003C_H$
RST 6.5	0034_H
RST 5.5	$002C_H$
TRAP	0024_H

The software interrupt instructions are included at the appropriate (or required) place in the main program. When the processor encounters the software instruction, it pushes the content of PC (Program Counter) to stack. Then loads the Vector address in PC and starts executing the Interrupt Service Routine (ISR) stored in this vector address. At the end of ISR, a return instruction - RET will be placed. When the RET instruction is executed, the processor POP the content of stack to PC. Hence the processor control returns to the main program after servicing the interrupt. Execution of ISR is referred to as servicing of interrupt. All software interrupts of 8085 are vectored interrupts. The software interrupts cannot be masked and they cannot be disabled. The software interrupts are RST0, RST1, ... RST7 (8 Nos).

HARDWARE INTERRUPTS OF 8085

These are the interrupts provided as signals to the microprocessor. There are five interrupt signals in 8085. They are Trap, RST 7.5, RST 6.5, RST 5.5 and INTR. The

priority of the interrupts is from TRAP to INTR. The program executed for the service of the interrupting device is called the service routine.

TRAP

• This interrupt is a Non-Maskable interrupt (NMI). It is unaffected by any mask or interrupt enable.

• TRAP is the highest priority and vectored interrupt(as vector address is fixed i.e. memory location where to transfer control).

• 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)

RST 7.5

• The RST 7.5 interrupt is a Maskable interrupt.

• It has the second highest priority.

• It is edge sensitive. i.e. 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.

RST 6.5 and 5.5

• The RST 6.5 and RST 5.5 both are level triggered (i.e.) 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.

These interrupts are classified further into two classes based on the destination address and response. Based on the service routine address, interrupts are classified in to vectored and non-vectored interrupt.

Vectored Interrupt:

If the address of the service routine is known to the microprocessor, i.e. if the service routine begins at a predefined address, then the interrupts are called vectored interrupts. The vectored address is calculated as $(nx8)_{16}$ where n is the number of RST.

For example:

The vectored address of RST 7.5 is 7.5 x 8=60.0

60 in hexadecimal number system is 003C. Therefore the branching address of RST 7.5 is 003C.

Interrupt	Address
RST 7.5	003C
RST 6.5	0034
RST 5.5	002C
TRAP (RST 4.5)	0024

Non vectored Interrupt:

The address of the service routine is not known in prior to the microprocessor. It is sent by the interrupting device.

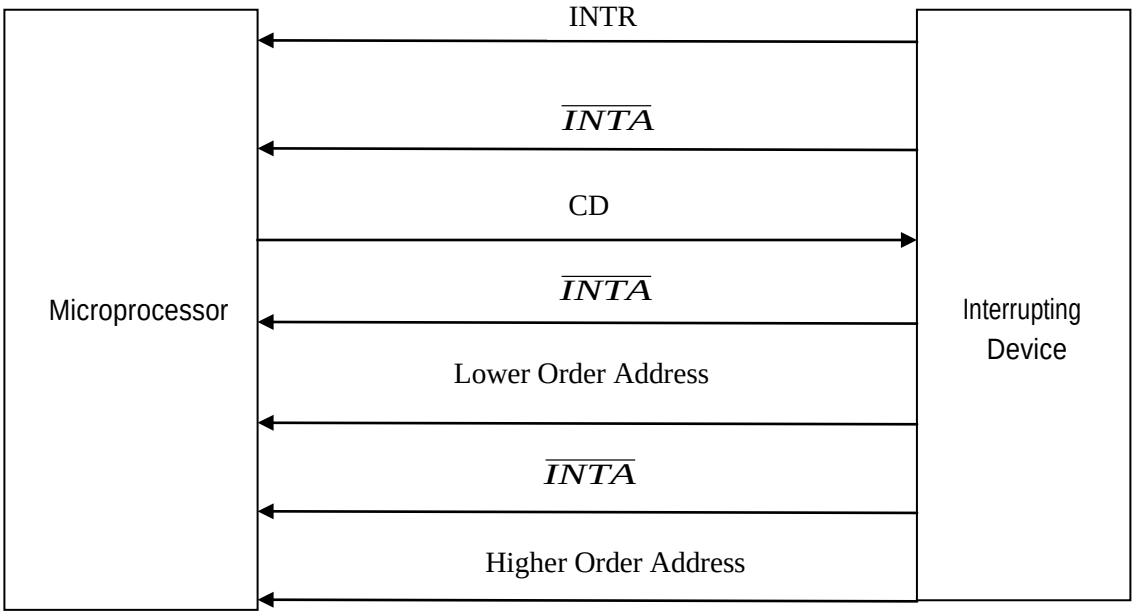


Fig 1.16.Work flow of Non vectored Interrupt

When the interrupt flipflop is enabled and INTR is high, microprocessor executes the current instruction and makes INTA low. The interrupting device sends the service routine address to the microprocessor as shown in fig 16.

Based on the flexibility to enable or disable interrupt, the interrupts are classified as maskable interrupt and non maskable interrupt.

Maskable Interrupt: Even if the interrupt signals are high, microprocessor will respond to these signals only when interrupt flip flop is enabled. Example RST 7.5, RST 6.5, RST 5.5, INTR

Non-Maskable Interrupt:

Once the signal is enabled, the microprocessor immediately responds to this interrupt. Example: TRAP

STACK

Stack is the upper part of the memory used for storing temporary information. It is a Last In First Out Memory (LIFO) . In 8085, it is accessed using PUSH and POP instructions. During pushing, the stack operates in a ¯decrement then store‖ style. The stack pointer is decremented first, then the information is placed on the stack. During poping, the stack operates in a ¯use then increment‖ style. The information is retrieved from the top of the the stack and then the pointer is incremented. The SP pointer always points to ¯the top of the stack‖.

Program Status Word (PSW)

The 8085 recognizes one additional register pair called the PSW (Program Status Word). This register pair is made up of the Accumulator and the Flags registers. It is possible to push the PSW onto the stack, do whatever operations are needed, then POP it off of the stack. The result is that the contents of the Accumulator and the status of the Flags are returned to what they were before the operations were executed.

Write an ALP to alter the contents of the flag register using STACK:

```
LXI SP 0000
PUSH PSW
POP H
MOV A L
MOV L A
PUSH H
```

POP PSW

Subroutines

A subroutine is a group of instructions that will be used repeatedly in different locations of the program. Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations. In Assembly language, a subroutine can exist anywhere in the code. However, it is customary to place subroutines separately from the main program. The 8085 has two instructions for dealing with subroutines. The CALL instruction is used to redirect program execution to the subroutine. The RET instruction is used to return the execution to the calling routine.

CALL

CALL 4000H (3 byte instruction)

When CALL instruction is fetched, the MP knows that the next two Memory location contains 16bit subroutine address in the memory.

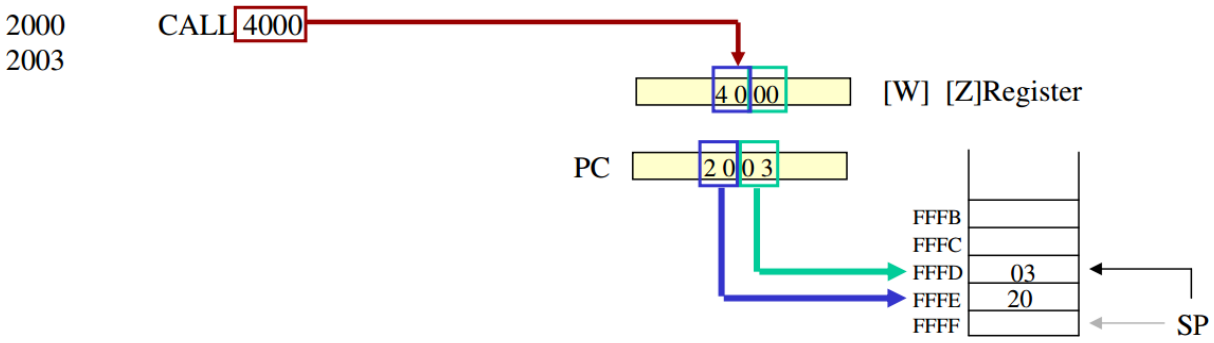


Fig 1.17.Work flow of CALL instruction

MP Reads the subroutine address from the next two memory location and stores the higher order 8bit of the address in the W register and stores the lower order 8bit of the address in the Z register. Push the address of the instruction immediately following the CALL onto the stack [Return address]. Loads the program counter with the 16-bit address supplied with the CALL instruction from WZ register as shown in fig 17.

RET

RET (1 byte instruction)

Retrieve the return address from the top of the stack. Load the program counter with the return address as seen in fig 18.

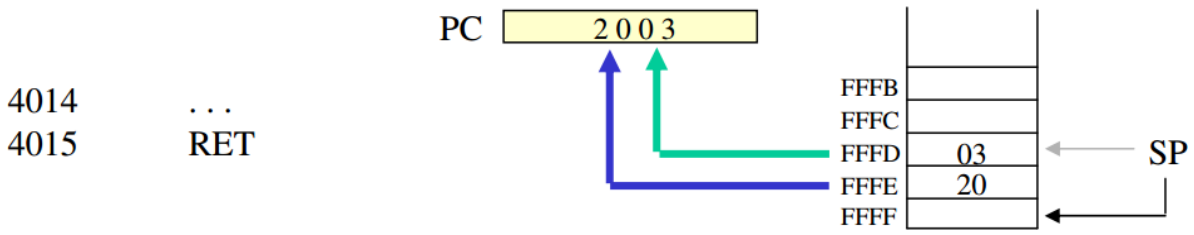


Fig1.18.Work flow of RET instruction

Write an ALP to ON and OFF 8 LEDs connected to 8085 MP.

```
START: MVI A,FF
       OUT 00
       CALL DELAY
       MVI A,00
       OUT 00
       CALL DELAY
       JUMP START


DELAY:  MVI  B,  10H  ;  7  T-States
LOOP2:  MVI  C,  FFH  ;  7  T-States
LOOP1: DCR C ; 4 T-States
       JNZ LOOP1   ; 10 T-States
        DCR B      ; 4 T-States
       JNZ LOOP2   ; 10 T-States
       RET
```

The calculation remains the same except that it the formula must be applied recursively to each loop. Start with the inner loop, then plug that delay in the calculation of the outer loop.

•Delay of inner loop

-T_{O1} = 7 T-States

•MVI C, FFH instruction

-T_{L1} = (255 X 14) - 3 = 3567 T-States

•14 T-States for the DCR C and JNZ instructions repeated 255 times (FF_{16} = 255_{10}) minus 3 for the final JNZ.

-T_{LOOP1} = 7 + 3567 = 3574 T-States

•Delay of outer loop

-T_{O2} = 7 T-States

•MVI B, 10H instruction

-T_{L1} = (16 X (14 + 3574)) - 3 = 57405 T-States

•14 T-States for the DCR B and JNZ instructions and 3574 T-States for loop1 repeated 16 times (10_{16} = 16_{10}) minus 3 for the final JNZ.

-T_{Delay} = 7 + 57405 = 57412 T-States

•Total Delay

-T_{Delay} = 57412 X 0.5 ▪Sec= 28.706 mSec

COMPARISION of 8085 with Z80 and MC6800 Microprocessor

S.No.	8085 Microprocessor	Z80 Microprocessor
1	Data Lines are MULTIPLEXED	It has no MULTIPLEXED lines
2	74 instructions	158 Instructions
3	Operates at 3 to 5MHz	Operates at 4 to 20 MHz
4	It has 5 interrupts	It has two interrupts
5	No on board dynamic memory	It has on board logic to refresh Dynamic memory

6	It contains no Index register	It has two Index register
7	It contains SIM & RIM	It contains no SIM & RIM

S.No.	8085 Microprocessor	MC6800 Microprocessor
1	It operates on Clock frequency of 3 to 5 MHz.	It operates at 1 MHz frequency.
2	8085 has no Index register.	It has one index register.
3	8085 has on board clock logic circuit.	No clock logic circuit.
4	8085 has one Accumulator Register.	MC6800 has two Accumulator Registers.
5	8085 has five interrupts.	MC 6800 have two interrupts.
6	It has total 674 Instructions.	MC6800 has total 72 instructions.

8085 PROGRAMS

I. 8-BIT ADDITION.

Program: **Function**

MVI C, 00 - Clear one register for carry (Reg C)

LDA 9100 -Load the accumulator with the first data

MOV B, A - Move the accumulator content to one register (Reg B)

LDA 9101 -Load the accumulator with the second data

ADD B - Add Reg B content to accumulator

JNC L1: -Check for carry, if there is no carry, go to step 8

INRC - Increment reg C to indicate carry

L1 :STA 9200 - Store the results

MOV A, C - carry in Reg C

STA 9201 - sum in accumulator to memory locations

RST 1 -stop

Sample Data:

Input	Output
9100 – 04	9200 – 0C
9101 – 08	9201 – 00
9100 – FF	9200 – FE
9101 – FF	9201 – 01

II. 8-BIT SUBTRACTION

Program: Function

MVIC, 00 - Clear one register for borrow (Reg C)

LDA 9200 -Load the accumulator with the first data

MOV B, A - Move the accumulator content to one register (Reg B)

LDA 9201 -Load the accumulator with the second data

SUB B - Subtract Reg B content from accumulator content

JNC L1: -Check for carry, if there is no carry, go to step 10

CMA -Complement accumulator content

INR A -increment accumulator content

INRC - Increment reg C to indicate borrow

L1 :STA 9200 - Store the results

MOV A, C - borrow in Reg C

STA 9201 - difference in accumulator to memory locations

RST 1 -stop

Sample Data:

Input	Output
9200 – FF	9200 – 55
9201 – AA	9201 – 01
9200 – BB	9200 – 44
9201 – FF	9201 – 00

III. 16-BIT ADDITION

Program:		Function
MVIC, 00-		Clear one register for carry (Reg C)
LHLD	9100	-Load the first data in HL register pair
XCHG		-Swap the contents of HL and DE pairs
LHLD		-Load the second data in HL register pair
DAD	D	-Double add the contents of HL and DE pairs
JNC	L1	- Check for carry, if there is no carry go to step 8
INRC		- Increment Reg C
L1 :SHLD	9104	- Store the result which is in HL pair in a memory location
MOV	A, C	- Move the carry in Reg C to accumulator
STA	9106	- Store the accumulator content in memory
1. RST	1	- Stop

Sample Data:

Input	Output
9100 – 06	9104 – 09
9101 – 05	9105 – 0B
9102 – 03	9106 – 00

9103 – 06

9100 – 06	9104 – 09
9101 – F0	9105 – E0
9102 – 03	9106 – 01
9103 – F0	

IV. REVERSE THE STRING

Program: **Function**

MVIB, 06- Initialize one register (Reg B) with the length of the string

LXI H, 9100 - Initialize one register pair (HL) with the starting address of the source array

LXI D, 9205 - Initialize one register pair (DE) with the ending address of the destination array

L1 :MOV A, M - Move the memory content to accumulator

STAX D - Store the accumulator content in DE pair

INX H - Increment HL pair

DCX D - Decrement DE pair

DCR B - Decrement the counter register – Reg B

JNZ L1 - Check for zero, if not zero, go to step 4

RST1 - Stop

Sample Data:

Input	**Output**
9100 – 0E	9200 – 0C
9101 – 0E	9201 – 00
9102 – 0F	9202 – 0F

9103 – 0F	9203 – 0F
9104 – 00	9204 – 0E
9105 – 0C	9205 – 0E

V. FACTORIAL OF A NUMBER

Program: **Function**

LDA 9100 -Load the accumulator with the given data
MOV B, A -Move the accumulator content to a register (Reg B)
MOV C, A -Move the accumulator content to another register
(Reg C)DCR C -Decrement Reg C
L2 : MOV D, C -Move the content of Reg C to
Reg DMVI A, 00 -Clear the accumulator
content
L1 : ADD B -Add Reg B content to
accumulatorDCR D -Decrement Reg D
JNZ L1 -Check for zero, if not zero, go
to step 7MOV B, A -Move accumulator content
to Reg B DCR C -Decrement Reg C
JNZ L2 -Check for zero, if not zero, go to step 5
STA 9101 -Store the accumulator content in a memory
addressRST 1 -Stop

Sample Data:

Input	**Output**
9100 – 04	9101 – 18

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

SCHOOL OF ELECTRICAL AND ELECTRONICS
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

UNIT 2
Microprocessor , Interfacing and Its Applications-SECA1508

INTEL 8086

Features of 8086 Microprocessor:

- Intel 8086 was launched in 1978.
- It was the first 16-bit microprocessor.
- This microprocessor had major improvement over the execution speed of 8085.
- It is available as 40-pin Dual-Inline-Package (DIP).
- It is available in three versions:
  a. 8086 (5 MHz)
  b. 8086-2 (8 MHz)
  c. 8086-1 (10 MHz)
- It consists of 29,000 transistors.
- It has a 16 line data bus and 20 line address bus.
- It could address up to 1 MB of memory.
- It has more than 20,000 instructions.

PIN DIAGRAM OF 8086

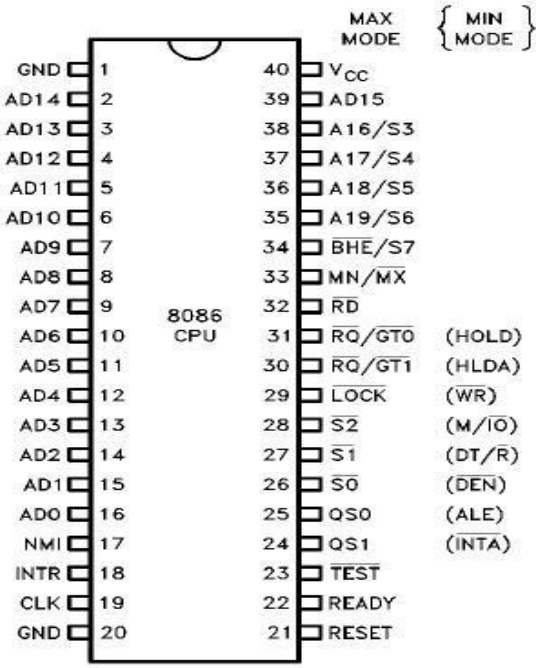


FIG 2.1 PIN DIAGRAM OF 8086

- **AD_0-AD_{15}**

These lines are multiplexed bidirectional address/data bus. During T1, they carry lower order 16-bit address. In the remaining clock cycles,they carry 16-bit data. AD0-AD7 carry lower order byte of data. AD8-AD15 carry higher order byte of data.

- **A19/S6, A18/S5, A17/S4, A16/S3** [Pin 35-38 (Unidirectional)]

These lines are multiplexed unidirectional address and status bus. During T1, they carry higher order 4-bit address. In the remaining clock cycles, they carry status signals.

- **BHE / S7** [Pin 34 (Output)]

BHE stands for Bus High Enable. BHE signal is used to indicate the transfer of data over higher order data bus (D8 – D15). 8-bit I/O devices use this signal. It is multiplexed with status pin S7.

- **RD (Read)** [Pin 32 (Output)]

It is a read signal used for read operation. It is an output signal. It is an active low signal.

- **READY** [Pin 22 (Input)]

This is an acknowledgement signal from slower I/O devices or memory. It is an active high signal. When high, it indicates that the device is ready to transfer data. When low, then microprocessor is in wait state.

- **RESET**[Pin 21 (Input)]

It is a system reset. It is an active high signal. When high, microprocessor enters into reset state and terminates the current activity. It must be active for atleast four clock cycles to reset the microprocessor.

- **INTR** [Pin 18 (Input)]

It is an interrupt request signal. It is active high. It is level triggered

- **NMI**[Pin 17 (Input)]

It is a non-maskable interrupt signal. It is an active high. It is an edge triggeredinterrupt.

- **TEST**[Pin 23 (Input)]

It is used to test the status of math coprocessor 8087. The BUSY pin of 8087 isconnected to this pin of 8086. If low, execution continues else microprocessor is in wait state.

- **CLK**[Pin 19 (Input)]

This clock input provides the basic timing for processor operation. It is symmetric square wave with 33% duty cycle. The range of frequency of different versions is 5 MHz, 8 MHz and 10 MHz.

- **VCC and VSS**[Pin 40 and Pin 20 (Input)]

VCC is power supply signal. +5V DC is supplied through this pin. VSS is ground signal

- **MN / MX**[Pin 33 (Input)]

8086 works in two modes: Minimum Mode, Maximum Mode. If MN/MX is high, it worksin minimum mode. If MN/MX is low, it works in maximum mode. Pins 24 to 31 issue twodifferent sets of signals. One set of signals is issued when CPU operates in minimum mode. Other set of signals isissued when CPU operates in maximum mode.

PIN Description for Minimum Mode

- **INTA [**Pin 24 (Output)]
  This is an interruptacknowledge signal. When microprocessorreceives INTR signal, it acknowledges the interrupt by generating this signal. It is an active low signal

- **ALE [**Pin 25 (Output)].
  This is an Address LatchEnable signal. It indicates that valid address is available on bus AD0 – AD15. It is an active high signal and remains high during T1 state. It is connected to enable pin of latch 8282.

- **DEN**[Pin 26 (Output)]
  This is a Data Enable signal. This signal is used toenable the transceiver 8286. Transceiver is used to separate the data from the address/data bus. It is an active low signal

- **DT / R**[Pin 27 (Output)]
  This is a DataTransmit/Receive signal. It decides the direction ofdata flow through the transceiver. When it is high, data is transmitted out. When it is low, data is received in.

- **M / IO**[Pin 28 (Output)]
  This signal is issued by the microprocessor to distinguish memory access from I/O access. When it is high, memory is accessed. When it is low, I/O devices are accessed.

- **WR [**Pin 29 (Output)]
  It is a Write signal. It is used to write data inmemory or output device depending on the status of M/IO signal. It is an active low signal

- **HLDA**[Pin 30 (Output)]
  It is a Hold Acknowledge signal. It is issued after receiving the HOLD signal. Itis an active high signal

- **HOLD**[Pin 31 (Input)]
  When DMA controller needs to use address/data bus, it sends a request to the CPU through this pin. It is an active high signal. When microprocessor receives HOLD signal, it issues HLDA signal to the DMA controller.

PIN Description for Maximum Mode

- **QS1 and QS0** [Pin 24 and 25 (Output)]

These pins provide the status of instruction queue.

QS_1	QS_0	STATUS
0	0	NO OPERATION
0	1	1st byte of opcode from queue
1	0	Empty queue
1	1	Subsequent byte from queue

- **S0, S1, S2** [Pin 26, 27, 28 (Output)]

These status signals indicate the operation being done by the microprocessor. This information is required by the Bus Controller 8288. Bus controller 8288 generates all memory and I/O control signals.

S_0	S_1	S_2	STATUS
0	0	0	Interrupt Acknowledge
0	0	1	I/O Read
0	1	0	I/O Write
0	1	1	Halt
1	0	0	Opcode Fetch
1	0	1	Memory Read
1	1	0	Memory Write
1	1	1	Passive

- **LOCK**[Pin 29 (Output)]

This signal indicates that other processors should not ask CPU to relinquish the system bus. When it goes low, all interrupts are masked and HOLD request is not granted. This pin is activated by using LOCK prefix on any instruction.

- **RQ/GT1 and RQ/GT0** [Pin 30 and 31 (Bi-directional)]

These are Request/Grant pins. Other processors request the CPU through these lines to release the system bus. After receiving the request, CPU sends acknowledge signal on the same lines. RQ/GT0 has higher priority than RQ/GT1.

Architecture of 8086

The microprocessors functions as the CPU in the stored program model of the digital computer. Its job is to generate all system timing signals and synchronize the transfer of data between memory, I/O, and itself. The microprocessor also has a S/W function. It must recognize, decode, and execute program instructions fetched from the memory unit. This requires an Arithmetic-Logic Unit (ALU) within the CPU to perform arithmetic and logical (AND, OR, NOT, compare, etc) functions.

The 8086 has pipelined architecture. The 8086 CPU is organized as two separate processors, called the Bus Interface Unit (BIU) and the Execution Unit (EU).

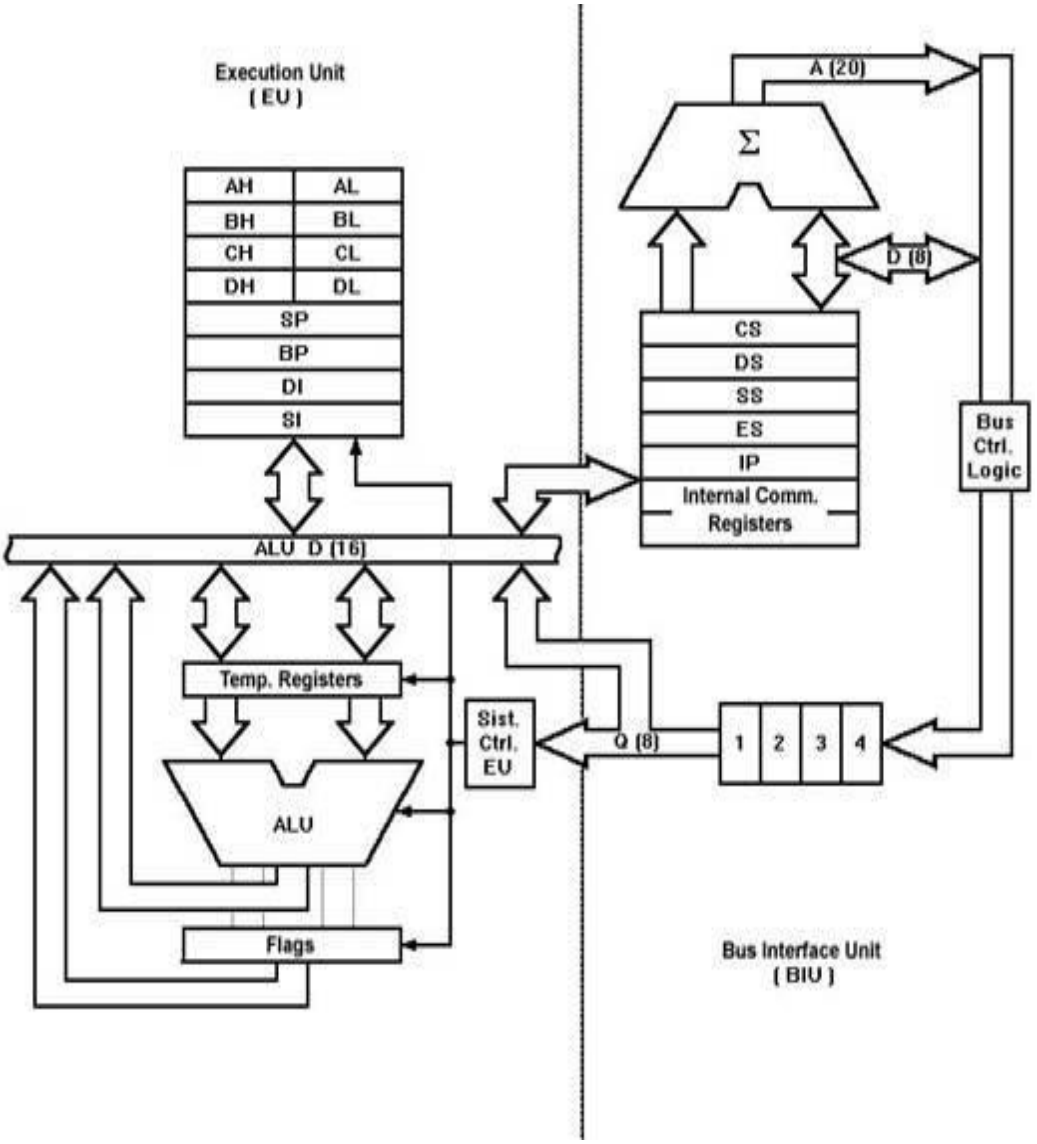


Fig 2.2 Architecture of 8086

A : Address Bus

D :Databus

Q : Q Bus

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.

Instruction Queue

1. To increase the execution speed, BIU fetches as many as six instruction bytes ahead to time from 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.

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.

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.

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:

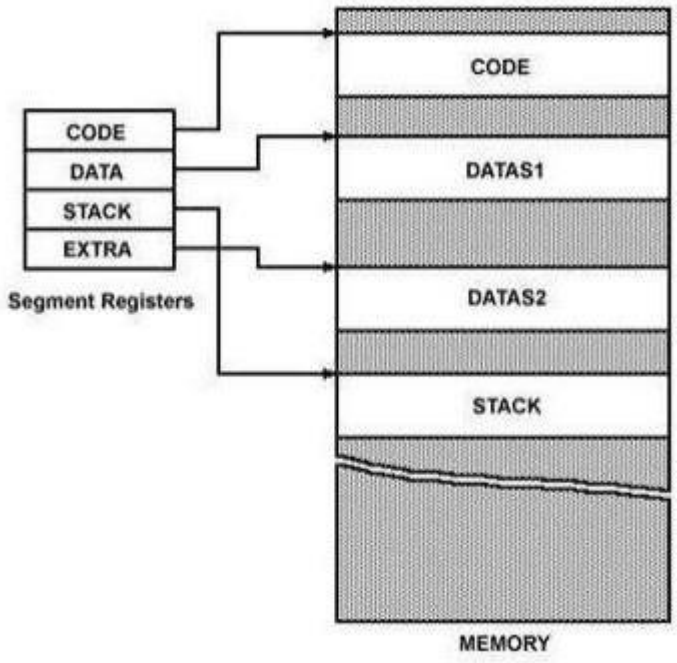


Fig 2.3: Memory Segments of 8086

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.

Flag Registers of 8086

Flag register in EU is of 16-bit , is shown

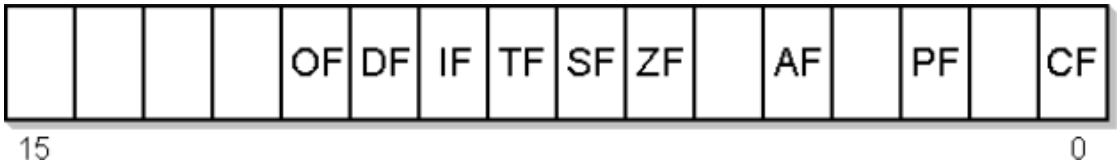


Fig 2.4 : Flag Register of 8086

Flag 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

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. D0 – D3) to upper nibble (i.e. D4 – D7), the AF flag is set i.e. carry given by D3 bit to D4 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.

Addressing modes of 8086

I. Addressing modes for register and immediate data

1) Register Addressing mode
2) Immediate Addressing mode

II. Addressing modes for memory data

3) Register Indirect Addressing mode
4) Direct Addressing mode
5) Based Addressing mode
6) Indexed Addressing mode
7) Base Relative Addressing mode
8) Base Indexed Addressing mode
9) String Addressing Mode

III. Addrressing modes for I/O port

10) Direct I/O port Addressing
11) Indirect I/O port Addressing

IV. Relative Addressing mode

12) Relative Addressing

V. Implied Addressing Mode

13) Implied Addressing

Register Addressing Mode

Data transfer using registers is called register addressing mode. Here operand value is present in register. For example
MOV AL,BL;
MOV AX,BX;

Immediate Addressing mode

When data is stored in code segment instead of data segment immediate addressing mode is used. Here operand value is present in the instruction. For example
MOV AX, 0A9FH;

Direct Addressing mode

When direct memory address is supplied as part of the instruction is called direct addressing mode. Operand offset value with respect to data segment is given in instruction. For example
MOV AX, [089DH];
ADD AX, [0ADH];

Register Indirect Addressing mode

Here operand offset is given in a cpu register. Register used are BX, SI(source index), DI(destination index), or BP(base pointer). BP holds offset w.r.t Stack segment, but SI,DI and BX refer to data segment. For example-
MOV [BX],AX;
ADD AX, [SI];

Based addressing Mode

In this mode EA is obtained by adding a displacement (signed 8 bit or unsigned 16 bit) value to the contents of BX or BP.The segment registers used are DS & SS.

When Memory is accessed,the 20 bit physical address is computed from BX and DS.On the other hand,when the stack is accessed,the 20 bit physical address is computed from BP and SS.

eg:- MOV AL,START[BX] or MOV AL,[BX+START]

Where START=02H(8 bit displacement),BX=2000H

Now the 20 bit Physical address is computed from DS and EA

If [DS]=5004(16),then

2000H	**BX**	**5**	**0**	**0**	**4**		
02H	**Displacement**		**2**	**0**	**0**	**2**	**EA**
2002H	**EA**	**5**	**2**	**0**	**4**	**0**	**20 bit Physical Address**

Here the source operand is in based Addressing Mode.EA is obtained by adding the value of START and [BX].The 20 bit physical address is produced from DS and EA.The 8 bit content of this memory location is moved to AL register.

Therefore the contents of the memory location 52042 is moved in to AL register.

Indexed Addressing mode

In this mode,the EA is calculated by adding the unsigned 16 bit or signed extended 8 bit displacement and the contents of SI or DI.

eg:- MOV BH,START[SI]

Moves the contents of the 20 bit address computed from the displacement START,SI and DS into BH register.The 8 bit displacement is provided by the programmer using the assembler pseudoindtruction such as EQU.For 16 bit displacement,the EU adds this to SI to determine EA.On the other hand,for 8 bit displacement the EU sign extends it to 16 bits and then adds to SI for determining EA.

Base Relative Addressing mode

Operand offset given by a sum of a value held either in BP, or BX and a constant offset specified as an operand. For example
MOV AX,[BP+1];
JMP [BX+1];

Base Indexed Addressing mode

Here operand offset is given by sum of either BX or BP with either SI or DI. For example
MOV AX, [BX+SI]
JMP [BP+DI]

String Addressing

This mode uses index registers.The string instructions automatically assume SI to point to the first byte or word of the source operand and DI to point to the first byte or word of the destination operand.

The segment register for the source is DS and may be overridden.The segment register for the destination must be ES and cannot be overridden.

The contents of SI and DI are automatically incremented by clearing DF (Direction Flag) to 0 by CLD instruction or automatically decremented by setting DF to 1 by STD instruction.

Direct Addressing Modes

Here the port number is a 8 bit immediate operand.This allows fixed access to ports numbered 0 to 255.

eg:- OUT 05H,AL

outputs [AL] to 8 bit port 05H

Indirect Addressing Mode

The port number is taken from DX allowing 64K 8 bit ports or 32K 16 bit ports.

eg:- IN AX,DX

If [DX]=5040,Inputs the 8 bit content of port 5040 into AL and 5041 into AH.

Relative Addressing Mode

In this mode,the operand is specified as a signed 8 bit displacement,relative to PC(Program Counter).

eg:- JNC START

then,if carry=0,PC is loaded with current PC contents plus the 8 bit signed value of START,otherwise the next instruction is executed.

Implied Addressing Mode

Instructions using this mode have no operands.

eg:- CLC

this clears the carry flag to zero.

INSTRUCTION SET OF 8086

I. DATA TRANSFER INSTRUCTIONS

- **MOV – MOV Destination, Source**

The MOV instruction copies a word or byte of data from a specified source to a specified destination. The destination can be a register or a memory

location. The source can be a register, a memory location or an immediate number. The source and destination cannot both be memory locations. They must both be of the same type (bytes or words).MOV instruction does not affect any flag.

MOV CX, 037AH	Put immediate number 037AH to CX
MOV BL, [437AH]	Copy byte in DS at offset 437AH to BL
MOV AX, BX	Copy content of register BX to AX
MOV DL, [BX]	Copy byte from memory at [BX] to DL
MOV DS, BX	Copy word from BX to DS register
MOV RESULT [BP], AX	Copy AX to two memory locations; AL to the first location, AH to the second; EA of the first memory location is sum of the displacement represented by RESULTS and content of BP. Physical address = EA + SS.
MOV ES: RESULTS [BP], AX	Same as the above instruction, but physical address = EA + ES, because of the segment override prefix ES

- **XCHG – XCHG Destination, Source**

The XCHG instruction exchanges the content of a register with the content of another register or with the content of memory location(s). It cannot directly exchange the content of two memory locations. The source and destination must both be of the same type (bytes or words). The segment registers cannot be used in this instruction. This instruction does not affect any flag.

XCHG AX, DX	Exchange word in AX with word in DX
XCHG BL, CH	Exchange byte in BL with byte in CH
XCHG AL, PRICES [BX]	Exchange byte in AL with byte in memory at EA = PRICE [BX] in DS.

- **LEA – LEA Register, Source**

This instruction determines the offset of the variable or memory location named as the source and puts this offset in the indicated 16-bit register. LEA does not affect any flag.

LEA BX, PRICES	Load BX with offset of PRICE in DS

LEA BP, SS: STACK_TOP Load BP with offset of STACK_TOP in SS

LEA CX, [BX][DI] Load CX with EA = [BX] + [DI]

- **LDS – LDS Register, Memory address of the first word**

This instruction loads new values into the specified register and into the DS register from four successive memory locations. The word from two memory locations is copied into the specified register and the word from the next two memory locations is copied into the DS registers. LDS does not affect any flag.

LDS BX, [4326]	Copy content of memory at displacement 4326H in DS to BL, content of 4327H to BH. Copy content at displacement of 4328H and 4329H in DS to DS register.
LDS SI, SPTR	Copy content of memory at displacement SPTR and SPTR + 1 in DS to SI register. Copy content of memory at displacements SPTR + 2 and SPTR + 3 in DS to DS register. DS: SI now points at start of the desired string.

- **LES – LES Register, Memory address of the first word**

This instruction loads new values into the specified register and into the ES register from four successive memory locations. The word from the first two memory locations is copied into the specified register, and the word from the next two memory locations is copied into the ES register. LES does not affect any flag.

LES BX, [789AH]	Copy content of memory at displacement 789AH in DS to BL, content of 789BH to BH, content of memory at displacement 789CH and 789DH in DS is copied to ES register.
LES DI, [BX]	Copy content of memory at offset [BX] and offset [BX] + 1 in DS to DI register. Copy content of memory at offset [BX] + 2 and [BX] + 3 to ES register.

- **PUSH – PUSH Source**

The PUSH instruction decrements the stack pointer by 2 and copies a word from a specified source to the location in the stack segment to which the stack pointer points. The source of the word can be generalpurpose register, segment register, or memory. The stack segment register and the stack pointer must be initialized before this instruction can be used. PUSH can be used to save data on the stack so that it will not destroyed by a procedure. This instruction does not affect any flag.

- PUSH BX Decrement SP by 2, copy BX to stack.
- PUSH DS Decrement SP by 2, copy DS to stack.
- PUSH BL Illegal; must push a word
- PUSH TABLE [BX] Decrement SP by 2, and copy word from memory in DS at EA = TABLE + [BX] to stack

- **POP – POP Destination**

The POP instruction copies a word from the stack location pointed to by the stack pointer to a destination specified in the instruction. The destination can be a general-purpose register, a segment register or a memory location. The data in the stack is not changed. After the word is copied to the specified destination, the stack pointer is automatically incremented by 2 to point to the next word on the stack. The POP instruction does not affect any flag.

- POP DX Copy a word from top of stack to DX; increment SP by 2
- POP DS Copy a word from top of stack to DS; increment SP by 2
- POP TABLE [DX] Copy a word from top of stack to memory in DS with EA = TABLE + [BX]; increment SP by 2.

- **PUSHF (PUSH FLAG REGISTER TO STACK)**

The PUSHF instruction decrements the stack pointer by 2 and copies a word in the flag register to two memory locations in stack pointed to by the stack pointer. The stack segment register is not affected. This instruction does to affect any flag.

- **POPF (POP WORD FROM TOP OF STACK TO FLAG REGISTER)**

The POPF instruction copies a word from two memory locations at the top of the stack to the flag register and increments the stack pointer by 2. The stack segment register and word on the stack are not affected. This instruction does to affect any flag.

- **INPUT-OUTPUT INSTRUCTIONS**

- IN – IN Accumulator, Port

The IN instruction copies data from a port to the AL or AX register. If an 8-bit port is read, the data will go to AL. If a 16-bit port is read, the data will go to AX.

The IN instruction has two possible formats, fixed port and variable port. For fixed port type, the 8-bit address of a port is specified directly in the instruction. With this form, any one of 256 possible ports can be addressed.

- IN AL, OC8H Input a byte from port OC8H to AL
- IN AX, 34H Input a word from port 34H to AX

For the variable-port form of the IN instruction, the port address is loaded into the DX register before the IN instruction. Since DX is a 16-bit register, the port address can be any number between 0000H and FFFFH. Therefore, up to 65,536 ports are addressable in this mode.

- MOV DX, 0FF78H Initialize DX to point to port
  IN AL, DX Input a byte from 8-bit port 0FF78H to AL
  IN AX, DX Input a word from 16-bit port 0FF78H to AX

The variable-port IN instruction has advantage that the port address can be computed or dynamically determined in the program. Suppose, for example, that an 8086-based computer needs to input data from 10 terminals, each having its own port address. Instead of having a separate procedure to input data from each port, you can write one generalized input procedure and simply pass the address of the desired port to the procedure in DX.

The IN instruction does not change any flag.

- **OUT – OUT Port, Accumulator**

The OUT instruction copies a byte from AL or a word from AX to the specified port. The OUT instruction has two possible forms, fixed port and variable port.

For the fixed port form, the 8-bit port address is specified directly in the instruction. With this form, any one of 256 possible ports can be addressed.

- OUT 3BH, AL Copy the content of AL to port 3BH
- OUT 2CH, AX Copy the content of AX to port 2CH

For variable port form of the OUT instruction, the content of AL or AX will be copied to the port at an address contained in DX. Therefore, the DX register must be loaded with the desired port address before this form of the OUT instruction is used.

- MOV DX, 0FFF8H Load desired port address in DX
  OUT DX, AL Copy content of AL to port FFF8H
  OUT DX, AX Copy content of AX to port FFF8H

The OUT instruction does not affect any flag.

II. ARITHMETIC INSTRUCTIONS

ADD – ADD Destination, Source

ADC – ADC Destination, Source

These instructions add a number from some source to a number in some destination and put the result in the specified destination. The ADC also adds the status of the carry flag to the result. The source may be an immediate number, a register, or a memory location. The destination may be a register or a memory location. The source and the destination in an instruction cannot both be memory locations. The source and the destination must be of the same type (bytes or words). If you want to add a byte to a word, you must copy the byte to a word location and fill the upper byte of the word with 0's before adding. Flags affected: AF, CF, OF, SF, ZF.

ADD AL, 74H	Add immediate number 74H to content of AL. Result in AL
ADC CL, BL	Add content of BL plus carry status to content of CL
ADD DX, BX	Add content of BX to content of DX
ADD DX, [SI]	Add word from memory at offset [SI] in DS to content of DX
ADC AL, PRICES [BX]	Add byte from effective address [BX] plus carry status to content of AL
ADD AL, PRICES [BX]	Add content of memory at effective address PRICES [BX] to AL

SUB – SUB Destination, Source

SBB – SBB Destination, Source

These instructions subtract the number in some source from the number in some destination and put the result in the destination. The SBB instruction also subtracts the content of carry flag from the destination. The source may be an immediate number, a register or memory location. The destination can also be a register or a memory location. However, the source and the destination cannot both be memory location. The source and the destination must both be of the same type (bytes or words). If you want to subtract a byte from a word, you must first move the byte to a word location such as a 16-bit register and fill the upper byte of the word with 0's. Flags affected: AF, CF, OF, PF, SF, ZF.

SUB CX, BX CX – BX; Result in CX

SBB CH, AL Subtract content of AL and content of CF from content of CH. Result in CH

SUB AX, 3427H Subtract immediate number 3427H from AX

SBB BX, [3427H] Subtract word at displacement 3427H in DS and content of CF from BX

SUB PRICES [BX], 04H Subtract 04 from byte at effective address PRICES [BX], if PRICES is declared with DB; Subtract 04 from word at effective address PRICES [BX], if it is declared with DW.

SBB CX, TABLE [BX] Subtract word from effective address TABLE [BX] and status of CF from CX.

SBB TABLE [BX], CX Subtract CX and status of CF from word in memory at effective address TABLE[BX].

MUL – MUL Source

This instruction multiplies an unsigned byte in some source with an unsigned byte in AL register or an unsigned word in some source with an unsigned word in AX register. The source can be a register or a memory location. When a byte is multiplied by the content of AL, the result (product) is put in AX. When a word is multiplied by the content of AX, the result is put in DX and AX registers. If the most significant byte of a 16-bit result or the most significant word of a 32-bit result is 0, CF and OF will both be 0's. AF, PF, SF and ZF are undefined after a MUL instruction.

If you want to multiply a byte with a word, you must first move the byte to a word location such as an extended register and fill the upper byte of the word with all 0's. You cannot use the CBW instruction for this, because the CBW instruction fills the upper byte with copies of the most significant bit of the lower byte.

MUL BH Multiply AL with BH; result in AX

MUL CX Multiply AX with CX; result high word in DX, low word in AX

MUL BYTE PTR [BX] Multiply AL with byte in DS pointed to by [BX]

MUL FACTOR [BX] Multiply AL with byte at effective address FACTOR [BX], if it is declared as type byte with DB. Multiply AX with word at effective address FACTOR [BX], if it is declared as type word with DW.

MOV AX, MCAND_16 Load 16-bit multiplicand into AX MOV CL, MPLIER_8 Load 8-bit multiplier into CL MOV CH, 00H Set upper byte of CX to all 0's MUL CX AX times CX; 32-bit result in DX and AX

IMUL – IMUL Source

This instruction multiplies a signed byte from source with a signed byte in AL or a signed word from some source with a signed word in AX. The source can be a register or a memory location. When a byte from source is multiplied with content of AL, the signed result (product) will be put in AX. When a word from source is multiplied by AX, the result is put in DX and AX. If the magnitude of the product does not require all the bits of the destination, the unused byte / word will be filled with copies of the sign bit. If the upper byte of a 16-bit result or the upper word of a 32-bit result contains only copies of the sign bit (all 0's or all 1's), then CF and the OF will both be 0; If it contains a part of the product, CF and OF will both be 1. AF, PF, SF and ZF are undefined after IMUL.

If you want to multiply a signed byte with a signed word, you must first move the byte into a word location and fill the upper byte of the word with copies of the sign bit. If you move the byte into AL, you can use the CBW instruction to do this.

IMUL BH Multiply signed byte in AL with signed byte in BH; result in AX.

IMUL AX Multiply AX times AX; result in DX and AX

MOV CX, MULTIPLIER Load signed word in CX

MOV AL, MULTIPLICAND Load signed byte in AL

CBW Extend sign of AL into AH

IMUL CX Multiply CX with AX; Result in DX and AX

DIV – DIV Source

This instruction is used to divide an unsigned word by a byte or to divide an unsigned double word (32 bits) by a word. When a word is divided by a byte, the word must be in the AX register. The divisor can be in a register or a memory location. After the division, AL will contain the 8-bit quotient, and AH will contain the 8-bit remainder. When a double word is divided by a word, the most significant word of the double word must be in DX, and the least significant word of the double word must be in AX. After the division, AX will contain the 16-bit quotient and DX will contain the 16-bit remainder. If an attempt is made to divide by 0 or if the quotient is too large to fit in the destination (greater than FFH / FFFFH), the 8086 will generate a type 0 interrupt. All flags are undefined after a DIV instruction.

If you want to divide a byte by a byte, you must first put the dividend byte in AL and fill AH with all 0's. Likewise, if you want to divide a word by another word, then put the dividend word in AX and fill DX with all 0's.

DIV BL Divide word in AX by byte in BL; Quotient in AL, remainder in AH

DIV CX	Divide down word in DX and AX by word in CX; Quotient in AX, and remainder in DX.
DIV SCALE [BX]	AX / (byte at effective address SCALE [BX]) if SCALE [BX] is of type byte; or (DX and AX) / (word at effective address SCALE[BX] if SCALE[BX] is of type word

IDIV – IDIV Source

This instruction is used to divide a signed word by a signed byte, or to divide a signed double word by a signed word.

When dividing a signed word by a signed byte, the word must be in the AX register. The divisor can be in an 8-bit register or a memory location. After the division, AL will contain the signed quotient, and AH will contain the signed remainder. The sign of the remainder will be the same as the sign of the dividend. If an attempt is made to divide by 0, the quotient is greater than 127 (7FH) or less than –127 (81H), the 8086 will automatically generate a type 0 interrupt.

When dividing a signed double word by a signed word, the most significant word of the dividend (numerator) must be in the DX register, and the least significant word of the dividend must be in the AX register. The divisor can be in any other 16-bit register or memory location. After the division, AX will contain a signed 16-bit quotient, and DX will contain a signed 16-bit remainder. The sign of the remainder will be the same as the sign of the dividend. Again, if an attempt is made to divide by 0, the quotient is greater than +32,767 (7FFFH) or less than –32,767 (8001H), the 8086 will automatically generate a type 0 interrupt.

All flags are undefined after an IDIV.

If you want to divide a signed byte by a signed byte, you must first put the dividend byte in AL and signextend AL into AH. The CBW instruction can be used for this purpose. Likewise, if you want to divide a signed word by a signed word, you must put the dividend word in AX and extend the sign of AX to all the bits of DX. The CWD instruction can be used for this purpose.

IDIV BL	Signed word in AX/signed byte in BL
IDIV BP	Signed double word in DX and AX/signed word in BP
IDIV BYTE PTR [BX]	AX / byte at offset [BX] in DS

INC – INC Destination

The INC instruction adds 1 to a specified register or to a memory location. AF, OF, PF, SF, and ZF are updated, but CF is not affected. This means that if an 8-bit destination containing FFH or a 16-bit destination containing FFFFH is incremented, the result will be all 0's with no carry.

INC BL	Add 1 to contains of BL register
INC CX	Add 1 to contains of CX register
INC BYTE PTR [BX]	Increment byte in data segment at offset contained in BX.
INC WORD PTR [BX]	Increment the word at offset of [BX] and [BX + 1] in the data segment.
INC TEMP	Increment byte or word named TEMP in the data segment. Increment byte if MAX_TEMP declared with DB.Increment word if MAX_TEMP is declared with DW.
INC PRICES [BX]	Increment element pointed to by [BX] in array PRICES. Increment a word if PRICES is declared as an array of words; Increment a byte if PRICES is declared as an array of bytes.

DEC – DEC Destination

This instruction subtracts 1 from the destination word or byte. The destination can be a register or a memory location. AF, OF, SF, PF, and ZF are updated, but CF is not affected. This means that if an 8-bit destination containing 00H or a 16-bit destination containing 0000H is decremented, the result will be FFH or FFFFH with no carry (borrow).

DEC CL	Subtract 1 from content of CL register
DEC BP	Subtract 1 from content of BP register
DEC BYTE PTR [BX]	Subtract 1 from byte at offset [BX] in DS.
DEC WORD PTR [BP]	Subtract 1 from a word at offset [BP] in SS.
DEC COUNT	Subtract 1 from byte or word named COUNT in DS. Decrement a byte if COUNT is declared with a DB; Decrement a word if COUNT is declared with a DW.

DAA (DECIMAL ADJUST AFTER BCD ADDITION)

This instruction is used to make sure the result of adding two packed BCD numbers is adjusted to be a legal BCD number. The result of the addition must be in AL for DAA to work correctly. If the lower nibble in AL after an addition is greater than 9 or AF was set by the addition, then the DAA instruction will add 6 to the lower nibble in AL. If the result in the upper nibble of AL in now greater than 9 or if the carry flag was set by the addition or correction, then the DAA instruction will add 60H to AL.

Let AL = 59 BCD, and BL = 35 BCD

ADD AL, BL AL = 8EH; lower nibble > 9, add 06H to AL

DAA AL = 94 BCD, CF = 0

Let AL = 88 BCD, and BL = 49 BCD

ADD AL, BL AL = D1H; AF = 1, add 06H to AL

DAA AL = D7H; upper nibble > 9, add 60H to AL

AL = 37 BCD, CF = 1

The DAA instruction updates AF, CF, SF, PF, and ZF; but OF is undefined.

DAS (DECIMAL ADJUST AFTER BCD SUBTRACTION)

This instruction is used after subtracting one packed BCD number from another packed BCD number, to make sure the result is correct packed BCD. The result of the subtraction must be in AL for DAS to work correctly. If the lower nibble in AL after a subtraction is greater than 9 or the AF was set by the subtraction, then the DAS instruction will subtract 6 from the lower nibble AL. If the result in the upper nibble is now greater than 9 or if the carry flag was set, the DAS instruction will subtract 60 from AL.

Let AL = 86 BCD, and BH = 57 BCD

SUB AL, BH AL = 2FH; lower nibble > 9, subtract 06H from AL

AL = 29 BCD, CF = 0

Let AL = 49 BCD, and BH = 72 BCD

SUB AL, BH AL = D7H; upper nibble > 9, subtract 60H from AL

DAS AL = 77 BCD, CF = 1 (borrow is needed)

The DAS instruction updates AF, CF, SF, PF, and ZF; but OF is undefined.

CBW (CONVERT SIGNED BYTE TO SIGNED WORD)

This instruction copies the sign bit of the byte in AL to all the bits in AH. AH is then said to be the sign extension of AL. CBW does not affect any flag.

Let AX = 00000000 10011011 (-155 decimal)

CBW Convert signed byte in AL to signed word in AX
AX = 11111111 10011011 (-155 decimal)

CWD (CONVERT SIGNED WORD TO SIGNED DOUBLE WORD)

This instruction copies the sign bit of a word in AX to all the bits of the DX register. In other words, it extends the sign of AX into all of DX. CWD affects no flags.

Let DX = 00000000 00000000, and AX = 11110000 11000111 (–3897 decimal)

CWD	Convert signed word in AX to signed double word in DX:AX DX = 11111111 11111111 AX = 11110000 11000111 (–3897 decimal)

AAA (ASCII ADJUST FOR ADDITION)

Numerical data coming into a computer from a terminal is usually in ASCII code. In this code, the numbers 0 to 9 are represented by the ASCII codes 30H to 39H. The 8086 allows you to add the ASCII codes for two decimal digits without masking off the “3” in the upper nibble of each. After the addition, the AAA instruction is used to make sure the result is the correct unpacked BCD.

Let AL = 0011 0101 (ASCII 5), and BL = 0011 1001 (ASCII 9)

ADD AL, BL	AL = 0110 1110 (6EH, which is incorrect BCD)
AAA	AL = 0000 0100 (unpacked BCD 4) CF = 1 indicates answer is 14 decimal.

The AAA instruction works only on the AL register. The AAA instruction updates AF and CF; but OF, PF, SF and ZF are left undefined.

AAS (ASCII ADJUST FOR SUBTRACTION)

Numerical data coming into a computer from a terminal is usually in an ASCII code. In this code the numbers 0 to 9 are represented by the ASCII codes 30H to 39H. The 8086 allows you to subtract the ASCII codes for two decimal digits without masking the “3” in the upper nibble of each. The AAS instruction is then used to make sure the result is the correct unpacked BCD.

Let AL = 00111001 (39H or ASCII 9), and BL = 00110101 (35H or ASCII 5)

SUB AL, BL	AL = 00000100 (BCD 04), and CF = 0
AAS	AL = 00000100 (BCD 04), and CF = 0 (no borrow required)

Let AL = 00110101 (35H or ASCII 5), and BL = 00111001 (39H or ASCII 9)

SUB AL, BL	AL = 11111100 (- 4 in 2’s complement form), and CF = 1
AAS	AL = 00000100 (BCD 06), and CF = 1 (borrow required)

The AAS instruction works only on the AL register. It updates ZF and CF; but OF, PF, SF, AF are left undefined.

AAM (BCD ADJUST AFTER MULTIPLY)

Before you can multiply two ASCII digits, you must first mask the upper 4 bit of each. This leaves unpacked BCD (one BCD digit per byte) in each byte. After the two unpacked BCD digits are multiplied, the AAM instruction is used to adjust the product to two unpacked BCD digits in AX. AAM works only after the multiplication of two unpacked BCD bytes, and it works only the operand in AL. AAM updates PF, SF and ZF but AF; CF and OF are left undefined.

Let AL = 00000101 (unpacked BCD 5), and BH = 00001001 (unpacked BCD 9)

MUL BH AL x BH: AX = 00000000 00101101 = 002DH

AAM AX = 00000100 00000101 = 0405H (unpacked BCD for 45)

AAD (BCD-TO-BINARY CONVERT BEFORE DIVISION)

AAD converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL. This adjustment must be made before dividing the two unpacked BCD digits in AX by an unpacked BCD byte. After the BCD division, AL will contain the unpacked BCD quotient and AH will contain the unpacked BCD remainder. AAD updates PF, SF and ZF; AF, CF and OF are left undefined.

Let AX = 0607 (unpacked BCD for 67 decimal), and CH = 09H

AAD AX = 0043 (43H = 67 decimal)

DIV CH AL = 07; AH = 04; Flags undefined after DIV

If an attempt is made to divide by 0, the 8086 will generate a type 0 interrupt.

III. LOGICAL

INSTRUCTIONS AND –

AND Destination, Source

This instruction ANDs each bit in a source byte or word with the same numbered bit in a destination byte or word. The result is put in the specified destination. The content of the specified source is not changed.

The source can be an immediate number, the content of a register, or the content of a memory location. The destination can be a register or a memory location. The source and the destination cannot both be memory locations. CF and OF are both 0 after AND. PF, SF, and ZF are updated by the AND instruction. AF is

undefined. PF has meaning only for an 8-bit operand.

AND CX, [SI]	AND word in DS at offset [SI] with word in CX register; Result in CX register
AND BH, CL	AND byte in CL with byte in BH; Result in BH
AND BX, 00FFH	00FFH Masks upper byte, leaves lower byte unchanged.

OR – OR Destination, Source

This instruction ORs each bit in a source byte or word with the same numbered bit in a destination byte or word. The result is put in the specified destination. The content of the specified source is not changed.

The source can be an immediate number, the content of a register, or the content of a memory location. The destination can be a register or a memory location. The source and destination cannot both be memory locations. CF and OF are both 0 after OR. PF, SF, and ZF are updated by the OR instruction. AF is undefined. PF has meaning only for an 8-bit operand.

OR AH, CL	CL ORed with AH, result in AH, CL not changed
OR BP, SI	SI ORed with BP, result in BP, SI not changed
OR SI, BP	BP ORed with SI, result in SI, BP not changed
OR BL, 80H	BL ORed with immediate number 80H; sets MSB of BL to 1
OR CX, TABLE [SI]	CX ORed with word from effective address TABLE [SI]; Content of memory is not changed.

XOR – XOR Destination, Source

This instruction Exclusive-ORs each bit in a source byte or word with the same numbered bit in a destination byte or word. The result is put in the specified destination. The content of the specified source is not changed.

The source can be an immediate number, the content of a register, or the content of a memory location. The destination can be a register or a memory location. The source and destination cannot both be memory locations. CF and OF are both 0 after XOR. PF, SF, and ZF are updated. PF has meaning only for an 8-bit operand. AF is undefined.

XOR CL, BH	Byte in BH exclusive-ORed with byte in CL. Result in CL. BH not changed.
XOR BP, DI	Word in DI exclusive-ORed with word in BP. Result in BP. DI not changed.

XOR WORD PTR [BX], 00FFH Exclusive-OR immediate number 00FFH with word at offset [BX] in the data segment. Result in memory location [BX]

NOT – NOT Destination

The NOT instruction inverts each bit (forms the 1's complement) of a byte or word in the specified destination. The destination can be a register or a memory location. This instruction does not affect any flag.

NOT BX Complement content or BX register

NOT BYTE PTR [BX] Complement memory byte at offset [BX] in data segment.

NEG – NEG Destination

This instruction replaces the number in a destination with its 2's complement. The destination can be a register or a memory location. It gives the same result as the invert each bit and add one algorithm. The NEG instruction updates AF, AF, PF, ZF, and OF.

NEG AL Replace number in AL with its 2's complement

NEG BX Replace number in BX with its 2's complement

NEG BYTE PTR [BX] Replace byte at offset BX in DX with its 2's complement NEG WORD PTR [BP] Replace word at offset BP in SS with its 2's complement

CMP – CMP Destination, Source

This instruction compares a byte / word in the specified source with a byte / word in the specified destination. The source can be an immediate number, a register, or a memory location. The destination can be a register or a memory location. However, the source and the destination cannot both be memory locations. The comparison is actually done by subtracting the source byte or word from the destination byte or word. The source and the destination are not changed, but the flags are set to indicate the results of the comparison. AF, OF, SF, ZF, PF, and CF are updated by the CMP instruction. For the instruction CMP CX, BX, the values of CF, ZF, and SF will be as follows:

	CF	ZF	SF	
CX = BX		0	1	0 Result of subtraction is 0
CX > BX	0	0	0	No borrow required, so CF = 0
CX < BX	1	0	1	Subtraction requires borrow, so CF = 1

CMP AL, 01H Compare immediate number 01H with byte in AL

CMP BH, CL Compare byte in CL with byte in BH

CMP CX, TEMP Compare word in DS at displacement TEMP with word at CX CMP PRICES [BX], 49H Compare immediate number 49H with byte at offset [BX] in array PRICES

TEST – TEST Destination, Source

This instruction ANDs the byte / word in the specified source with the byte / word in the specified destination. Flags are updated, but neither operand is changed. The test instruction is often used to set flags before a Conditional jump instruction.

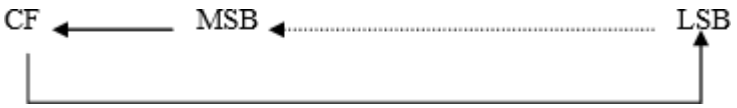
The source can be an immediate number, the content of a register, or the content of a memory location. The destination can be a register or a memory location. The source and the destination cannot both be memory locations. CF and OF are both 0's after TEST. PF, SF and ZF will be updated to show the results of the destination. AF is be undefined.

TEST AL, BH	AND BH with AL. No result stored; Update PF, SF, ZF.
TEST CX, 0001H	AND CX with immediate number 0001H; No result stored; Update PF, SF, ZF
TEST BP, [BX][DI]	AND word are offset [BX][DI] in DS with word in BP. No result stored. Update PF, SF, and ZF

IV. ROTATE AND SHIFT INSTRUCTION

RCL – RCL Destination, Count

This instruction rotates all the bits in a specified word or byte some number of bit positions to the left. The operation circular because the MSB of the operand is rotated into the carry flag and the bit in the carry flag is rotated around into LSB of the operand.



For multi-bit rotates, CF will contain the bit most recently rotated out of the MSB.

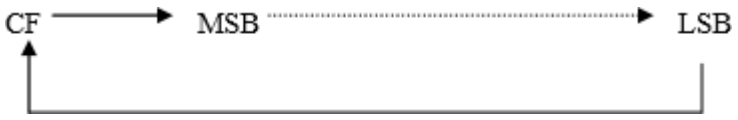
The destination can be a register or a memory location. If you want to rotate the operand by one bit position, you can specify this by putting a 1 in the count position of the instruction. To rotate by more than one bit position, load the desired number into the CL register and put “CL” in the count position of the instruction.

RCL affects only CF and OF. OF will be a 1 after a single bit RCL if the MSB was changed by the rotate. OF is undefined after the multi-bit rotate.

- RCL DX, 1 Word in DX 1 bit left, MSB to CF, CF to LSB
- MOV CL, 4 Load the number of bit positions to rotate into CL
  RCL SUM [BX], CL Rotate byte or word at effective address SUM [BX] 4 bits left Original bit 4 now in CF, original CF now in bit 3.

RCR – RCR Destination, Count

This instruction rotates all the bits in a specified word or byte some number of bit positions to the right. The operation circular because the LSB of the operand is rotated into the carry flag and the bit in the carry flag is rotate around into MSB of the operand.



For multi-bit rotate, CF will contain the bit most recently rotated out of the LSB.

The destination can be a register or a memory location. If you want to rotate the operand by one bit position, you can specify this by putting a 1 in the count position of the instruction. To rotate more than one bit position, load the desired number into the CL register and put “CL” in the count position of the instruction.

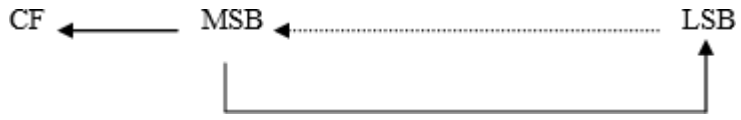
RCR affects only CF and OF. OF will be a 1 after a single bit RCR if the MSB was changed by the rotate. OF is undefined after the multi-bit rotate.

- RCR BX, 1 Word in BX right 1 bit, CF to MSB, LSB to CF
- MOV CL, 4 Load CL for rotating 4 bit position

RCR BYTE PTR [BX], 4 Rotate the byte at offset [BX] in DS 4 bit positions right CF = original bit 3, Bit 4 – original CF.

ROL – ROL Destination, Count

This instruction rotates all the bits in a specified word or byte to the left some number of bit positions. The data bit rotated out of MSB is circled back into the LSB. It is also copied into CF. In the case of multiple-bit rotate, CF will contain a copy of the bit most recently moved out of the MSB.



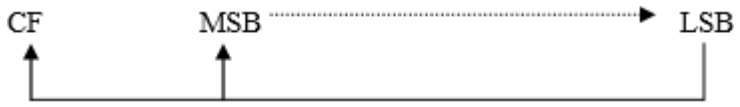
The destination can be a register or a memory location. If you to want rotate the operand by one bit position, you can specify this by putting 1 in the count position in the instruction. To rotate more than one bit position, load the desired number into the CL register and put “CL” in the count position of the instruction.

ROL affects only CF and OF. OF will be a 1 after a single bit ROL if the MSB was changed by the rotate.

- ROL AX, 1 Rotate the word in AX 1 bit position left, MSB to LSB and CF
- MOV CL, 04H Load number of bits to rotate in CL
  ROL BL, CL Rotate BL 4 bit positions
- ROL FACTOR [BX], 1 Rotate the word or byte in DS at EA = FACTOR [BX] by 1 bit position left into CF

ROR – ROR Destination, Count

This instruction rotates all the bits in a specified word or byte some number of bit positions to right. The operation is desired as a rotate rather than shift, because the bit moved out of the LSB is rotated around into the MSB. The data bit moved out of the LSB is also copied into CF. In the case of multiple bit rotates, CF will contain a copy of the bit most recently moved out of the LSB.



The destination can be a register or a memory location. If you want to rotate the operand by one bit position, you can specify this by putting 1 in the count position

in the instruction. To rotate by more than one bit position, load the desired number into the CL register and put “CL” in the count position of the instruction.

ROR affects only CF and OF. OF will be a 1 after a single bit ROR if the MSB was changed by the rotate.

- ROR BL, 1 Rotate all bits in BL right 1 bit position LSB to MSB and to CF
- MOV CL, 08H Load CL with number of bit positions to be rotated
  ROR WORD PTR [BX], CL Rotate word in DS at offset [BX] 8 bit position right

SAL – SAL Destination, Count SHL – SHL Destination, Count

SAL and SHL are two mnemonics for the same instruction. This instruction shifts each bit in the specified destination some number of bit positions to the left. As a bit is shifted out of the LSB operation, a 0 is put in the LSB position. The MSB will be shifted into CF. In the case of multi-bit shift, CF will contain the bit most recently shifted out from the MSB. Bits shifted into CF previously will be lost.

The destination operand can be a byte or a word. It can be in a register or in a memory location. If you want to shift the operand by one bit position, you can specify this by putting a 1 in the count position of the instruction. For shifts of more than 1 bit position, load the desired number of shifts into the CL register, and put “CL” in the count position of the instruction.

The flags are affected as follow: CF contains the bit most recently shifted out from MSB. For a count of one, OF will be 1 if CF and the current MSB are not the same. For multiple-bit shifts, OF is undefined. SF and ZF will be updated to reflect the condition of the destination. PF will have meaning only for an operand in AL. AF is undefined.

- SAL BX, 1 Shift word in BX 1 bit position left, 0 in LSB
- MOV CL, 02h Load desired number of shifts in CL
  SAL BP, CL Shift word in BP left CL bit positions, 0 in LSBs
- SAL BYTE PTR [BX], 1 Shift byte in DX at offset [BX] 1 bit position left, 0 in LSB

SAR – SAR Destination, Count

This instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted out of the MSB position, a copy of the old

MSB is put in the MSB position. In other words, the sign bit is copied into the MSB. The LSB will be shifted into CF. In the case of multiple-bit shift, CF will contain the bit most recently shifted out from the LSB. Bits shifted into CF previously will be lost.

The destination operand can be a byte or a word. It can be in a register or in a memory location. If you want to shift the operand by one bit position, you can specify this by putting a 1 in the count position of the instruction. For shifts of more than 1 bit position, load the desired number of shifts into the CL register, and put “CL” in the count position of the instruction.

The flags are affected as follow: CF contains the bit most recently shifted in from LSB. For a count of one, OF will be 1 if the two MSBs are not the same. After a multi-bit SAR, OF will be 0. SF and ZF will be updated to show the condition of the destination. PF will have meaning only for an 8- bit destination. AF will be undefined after SAR.

- SAR DX, 1 Shift word in DI one bit position right, new MSB = old MSB
- MOV CL, 02H Load desired number of shifts in CL
  SAR WORD PTR [BP], CL Shift word at offset [BP] in stack segment right by two bit positions, the two MSBs are now copies of original LSB

SHR – SHR Destination, Count

This instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted out of the MSB position, a 0 is put in its place. The bit shifted out of the LSB position goes to CF. In the case of multi-bit shifts, CF will contain the bit most recently shifted out from the LSB. Bits shifted into CF previously will be lost.

The destination operand can be a byte or a word in a register or in a memory location. If you want to shift the operand by one bit position, you can specify this by putting a 1 in the count position of the instruction. For shifts of more than 1 bit position, load the desired number of shifts into the CL register, and put “CL” in the count position of the instruction. The flags are affected by SHR as follow: CF contains the bit most recently shifted out from LSB. For a count of one, OF will be 1 if the two MSBs are not both 0’s. For multiple-bit shifts, OF will be meaningless. SF

and ZF will be updated to show the condition of the destination. PF will have meaning only for an 8-bit destination. AF is undefined.

- SHR BP, 1 Shift word in BP one bit position right, 0 in MSB
- MOV CL, 03H Load desired number of shifts into CL
  SHR BYTE PTR [BX] Shift byte in DS at offset [BX] 3 bits right; 0's in 3 MSBs

V. STRING MANIPULATION INSTRUCTIONS

MOVS – MOVS Destination String Name, Source String Name

MOVSB – MOVSB Destination String Name, Source StringName

MOVSW – MOVSW Destination String Name, Source String Name

This instruction copies a byte or a word from location in the data segment to a location in the extra segment. The offset of the source in the data segment must be in the SI register. The offset of the destination in the extra segment must be in the DI register. For multiple-byte or multiple-word moves, the number of elements to be moved is put in the CX register so that it can function as a counter. After the byte or a word is moved, SI and DI are automatically adjusted to point to the next source element and the next destination element. If DF is 0, then SI and DI will incremented by 1 after a byte move and by 2 after a word move. If DF is 1, then SI and DI will be decremented by 1 after a byte move and by 2 after a word move. MOVS does not affect any flag.

When using the MOVS instruction, you must in some way tell the assembler whether you want to move a string as bytes or as word. There are two ways to do this. The first way is to indicate the name of the source and destination strings in the instruction, as, for example. MOVS DEST, SRC. The assembler will code the instruction for a byte / word move if they were declared with a DB / DW. The second way is to add a "B" or a "W" to the MOVS mnemonic. MOVSB says move a string as bytes; MOVSW says move a string as words.

- MOV SI, OFFSET SOURCE Load offset of start of source string in DS into SI MOV DI, OFFSET DESTINATION Load offset of start of destination string in ES into DI CLD Clear DF to auto increment SI and DI after move MOV CX, 04H
  Load length of string into CX as counter
  REP MOVSB Move string byte until CX = 0

LODS / LODSB / LODSW (LOAD STRING BYTE INTO AL OR STRING WORDINTO AX)

This instruction copies a byte from a string location pointed to by SI to AL, or a word from a string location pointed to by SI to AX. If DF is 0, SI will be automatically incremented (by 1 for a byte string, and 2 for a word string) to point to the next element of the string. If DF is 1, SI will be automatically decremented (by 1 for a byte string, and 2 for a word string) to point to the previous element of the string. LODS does not affect any flag.

- CLD — Clear direction flag so that SI is auto-incremented
  MOV SI, OFFSET SOURCE — Point SI to start of string
  LODS SOURCE — Copy a byte or a word from string to AL or AX

Note: The assembler uses the name of the string to determine whether the string is of type bye or type word. Instead of using the string name to do this, you can use the mnemonic LODSB to tell the assembler that the string is type byte or the mnemonic LODSW to tell the assembler that the string is of type word.

STOS / STOSB / STOSW (STORE STRING BYTE OR STRING WORD)

This instruction copies a byte from AL or a word from AX to a memory location in the extra segment pointed to by DI. In effect, it replaces a string element with a byte from AL or a word from AX. After the copy, DI is automatically incremented or decremented to point to next or previous element of the string. If DF is cleared, then DI will automatically incremented by 1 for a byte string and by 2 for a word string. If DI is set, DI will be automatically decremented by 1 for a byte string and by 2 for a word string. STOS does not affect any flag.

- MOV DI, OFFSET TARGET
  STOS TARGET

Note: The assembler uses the string name to determine whether the string is of type byte or type word. If it is a byte string, then string byte is replaced with content of AL. If it is a word string, then string word is replaced with content of AX.

- MOV DI, OFFSET TARGET
  STOSB

“B” added to STOSB mnemonic tells assembler to replace byte in string with byte from AL. STOSW would tell assembler directly to replace a word in the string with a word from AX.

CMPS / CMPSB / CMPSW (COMPARE STRING BYTES OR STRING WORDS)

This instruction can be used to compare a byte / word in one string with a byte / word in another string. SI is used to hold the offset of the byte or word in the source string, and DI is used to hold the offset of the byte or word in the destination string.

The AF, CF, OF, PF, SF, and ZF flags are affected by the comparison, but the two operands are not affected. After the comparison, SI and DI will automatically be incremented or decremented to point to the next or previous element in the two strings. If DF is set, then SI and DI will automatically be decremented by 1 for a byte string and by 2 for a word string. If DF is reset, then SI and DI will automatically be incremented by 1 for byte strings and by 2 for word strings. The string pointed to by SI must be in the data segment. The string pointed to by DI must be in the extra segment.

The CMPS instruction can be used with a REPE or REPNE prefix to compare all the elements of a string.

- MOV SI, OFFSET FIRST — Point SI to source string
  MOV DI, OFFSET SECOND — Point DI to destination string
  CLD — DF cleared, SI and DI will auto-increment after compare
  MOV CX, 100 — Put number of string elements in CX
  REPE CMPSB — Repeat the comparison of string bytes until end of string or until compared bytes are not equal

CX functions as a counter, which the REPE prefix will cause CX to be decremented after each compare. The B attached to CMPS tells the assembler that the strings are of type byte. If you want to tell the assembler that strings are of type word, write the instruction as CMPSW. The REPE CMPSW instruction will cause the pointers in SI and DI to be incremented by 2 after each compare, if the direction flag is set.

SCAS / SCASB / SCASW (SCAN A STRING BYTE OR A STRING WORD)

SCAS compares a byte in AL or a word in AX with a byte or a word in ES pointed to by DI. Therefore, the string to be scanned must be in the extra segment, and DI must contain the offset of the byte or the word to be compared. If DF is cleared, then DI will be incremented by 1 for byte strings and by 2 for word strings. If DF is set, then DI will be decremented by 1 for byte strings and by 2 for word strings. SCAS affects AF, CF, OF, PF, SF, and ZF, but it does not change either the operand in AL (AX) or the operand in the string.

The following program segment scans a text string of 80 characters for a carriage return, 0DH, and puts the offset of string into DI:

- MOV DI, OFFSET STRING
  MOV AL, 0DH — Byte to be scanned for into AL

```
MOV CX, 80                  CX used as element counter
CLD                    Clear DF, so that DI auto increments
REPNE SCAS STRING           Compare byte in string with byte in AL
```

REP / REPE / REPZ / REPNE / REPNZ (PREFIX) (REPEAT STRINGINSTRUCTION UNTIL SPECIFIED CONDITIONS EXIST)

REP is a prefix, which is written before one of the string instructions. It will cause the CX register to be decremented and the string instruction to be repeated until CX = 0. The instruction REP MOVSB, for example, will continue to copy string bytes until the number of bytes loaded into CX has been copied.

REPE and REPZ are two mnemonics for the same prefix. They stand for repeat if equal and repeat if zero, respectively. They are often used with the Compare String instruction or with the Scan String instruction. They will cause the string instruction to be repeated as long as the compared bytes or words are equal (ZF = 1) and CX is not yet counted down to zero. In other words, there are two conditions that will stop the repetition: CX = 0 or string bytes or words not equal.

REPE CMPSB Compare string bytes until end of string or until string bytes not equal.

REPNE and REPNZ are also two mnemonics for the same prefix. They stand for repeat if not equal and repeat if not zero, respectively. They are often used with the Compare String instruction or with the Scan String instruction. They will cause the string instruction to be repeated as long as the compared bytes or words are not equal (ZF = 0) and CX is not yet counted down to zero.

REPNE SCASW Scan a string of word until a word in the string matches the word in AX or until all of the string has been scanned.

The string instruction used with the prefix determines which flags are affected.

VI. CONTROL TRANSFER INSTRUCTIONS

JMP (UNCONDITIONAL JUMP TO SPECIFIED DESTINATION)

This instruction will fetch the next instruction from the location specified in the instruction rather than from the next location after the JMP instruction. If the destination is in the same code segment as the JMP instruction, then only the instruction pointer will be changed to get the destination location. This is referred to as a near jump. If the destination for the jump instruction is in a segment with a name different from that of the segment containing the JMP instruction, then both the instruction pointer and the code segment register

content will be changed to get the destination location. This referred to as a far jump. The JMP instruction does not affect any flag.

JMP CONTINUE	This instruction fetches the next instruction from address at label CONTINUE. If the label is in the same segment, an offset coded as part of the instruction will be added to the instruction pointer to produce the new fetch address. If the label is another segment, then IP and CS will be replaced with value coded in part of the instruction. This type of jump is referred to as direct because the displacement of the destination or the destination itself is specified directly in the instruction.
JMP BX	This instruction replaces the content of IP with the content of BX. BX must first be loaded with the offset of the destination instruction in CS. This is a near jump. It is also referred to as an indirect jump because the new value of IP comes from a register rather than from the instruction itself, as in a direct jump.
JMP WORD PTR [BX]	This instruction replaces IP with word from a memory location pointed to by BX in DX. This is an indirect near jump.
JMP DWORD PTR [SI]	This instruction replaces IP with word pointed to by SI in DS. It replaces CS with a word pointed by SI + 2 in DS. This is an indirect far jump.

JAE / JNB / JNC (JUMP IF ABOVE OR EQUAL / JUMP IF NOT BELOW / JUMPIF NO CARRY)

If, after a compare or some other instructions which affect flags, the carry flag is 0, this instruction will cause execution to jump to a label given in the instruction. If CF is 1, the instruction will have no effect on program execution.

• CMP AX, 4371H	Compare (AX – 4371H)
JAE NEXT	Jump to label NEXT if AX above 4371H
• CMP AX, 4371H	Compare (AX – 4371H)
JNB NEXT	Jump to label NEXT if AX not below 4371H
• ADD AL, BL	Add two bytes
JNC NEXT	If the result with in acceptable range, continue

JB / JC / JNAE (JUMP IF BELOW / JUMP IF CARRY / JUMP IF NOT ABOVE OREQUAL)

If, after a compare or some other instructions which affect flags, the carry flag is a 1, this instruction will cause execution to jump to a label given in the instruction. If CF is 0, the instruction will have no effect on program execution.

- CMP AX, 4371H — Compare (AX – 4371H)
  JB NEXT — Jump to label NEXT if AX below 4371H
- ADD BX, CX — Add two words
  JC NEXT — Jump to label NEXT if CF = 1
- CMP AX, 4371H — Compare (AX – 4371H)
  JNAE NEXT — Jump to label NEXT if AX not above or equal to 4371H

JBE / JNA (JUMP IF BELOW OR EQUAL / JUMP IF NOT ABOVE)

If, after a compare or some other instructions which affect flags, either the zero flag or the carry flag is 1, this instruction will cause execution to jump to a label given in the instruction. If CF and ZF are both 0, the instruction will have no effect on program execution.

- CMP AX, 4371H — Compare (AX – 4371H)
  JBE NEXT — Jump to label NEXT if AX is below or equal to 4371H
- CMP AX, 4371H — Compare (AX – 4371H)
  JNA NEXT — Jump to label NEXT if AX not above 4371H

JG / JNLE (JUMP IF GREATER / JUMP IF NOT LESS THAN OR EQUAL)

This instruction is usually used after a Compare instruction. The instruction will cause a jump to the label given in the instruction, if the zero flag is 0 and the carry flag is the same as the overflow flag.

- CMP BL, 39H — Compare by subtracting 39H from BL
  JG NEXT — Jump to label NEXT if BL more positive than 39H
- CMP BL, 39H — Compare by subtracting 39H from BL
  JNLE NEXT — Jump to label NEXT if BL is not less than or equal to 39H

JGE / JNL (JUMP IF GREATER THAN OR EQUAL / JUMP IF NOT LESS THAN)

This instruction is usually used after a Compare instruction. The instruction will cause a jump to the label given in the instruction, if the sign flag is equal to the overflow flag.

- CMP BL, 39H Compare by subtracting 39H from BL
  JGE NEXT Jump to label NEXT if BL more positive than or equal to 39H
- CMP BL, 39H Compare by subtracting 39H from BL
  JNL NEXT Jump to label NEXT if BL not less than 39H

JLE / JNG (JUMP IF LESS THAN OR EQUAL / JUMP IF NOT GREATER)

This instruction is usually used after a Compare instruction. The instruction will cause a jump to the label given in the instruction if the zero flag is set, or if the sign flag not equal to the overflow flag.

- CMP BL, 39H Compare by subtracting 39H from BL
  JLE NEXT Jump to label NEXT if BL more negative than or equal to 39H
- CMP BL, 39H Compare by subtracting 39H from BL
  JNG NEXT Jump to label NEXT if BL not more positive than 39H

JE / JZ (JUMP IF EQUAL / JUMP IF ZERO)

This instruction is usually used after a Compare instruction. If the zero flag is set, then this instruction will cause a jump to the label given in the instruction.

- CMP BX, DX Compare (BX-DX)
  JE DONE Jump to DONE if BX = DX
- IN AL, 30H Read data from port 8FH
  SUB AL, 30H Subtract the minimum value.
  JZ START Jump to label START if the result of subtraction is 0

JNE / JNZ (JUMP NOT EQUAL / JUMP IF NOT ZERO)

This instruction is usually used after a Compare instruction. If the zero flag is 0, then this instruction will cause a jump to the label given in the instruction.

- IN AL, 0F8H Read data value from port
  CMP AL, 72 Compare (AL -72)
  JNE NEXT Jump to label NEXT if AL ≠ 72
- ADD AX, 0002H Add count factor 0002H to AX
  DEC BX Decrement BX
  JNZ NEXT Jump to label NEXT if BX ≠ 0

JS (JUMP IF SIGNED / JUMP IF NEGATIVE)

This instruction will cause a jump to the specified destination address if the sign flag is set. Since a 1 in the sign flag indicates a negative signed number, you can think of this instruction as saying “jump if negative”.

- ADD BL, DH Add signed byte in DH to signed byte in DL
  JS NEXT Jump to label NEXT if result of addition is negative number

JNS (JUMP IF NOT SIGNED / JUMP IF POSITIVE)

This instruction will cause a jump to the specified destination address if the sign flag is 0. Since a 0 in the sign flag indicate a positive signed number, you can think to this instruction as saying “jump if positive”.

- DEC AL Decrement AL
  JNS NEXT Jump to label NEXT if AL has not decremented to FFH

JP / JPE (JUMP IF PARITY / JUMP IF PARITY EVEN)

If the number of 1’s left in the lower 8 bits of a data word after an instruction which affects the parity flag is even, then the parity flag will be set. If the parity flag is set, the JP / JPE instruction will cause a jump to the specified destination address.

- IN AL, 0F8H Read ASCII character from Port F8H
  OR AL, AL Set flags
  JPE ERROR Odd parity expected, send error message if parity found even

JNP / JPO (JUMP IF NO PARITY / JUMP IF PARITY ODD)

If the number of 1’s left in the lower 8 bits of a data word after an instruction which affects the parity flag is odd, then the parity flag is 0. The JNP / JPO instruction will cause a jump to the specified destination address, if the parity flag is 0.

- IN AL, 0F8H Read ASCII character from Port F8H
  OR AL, AL Set flags
  JPO ERROR Even parity expected, send error message if parity found odd

JO (JUMP IF OVERFLOW)

The overflow flag will be set if the magnitude of the result produced by some signed arithmetic operation is too large to fit in the destination register or memory location.

The JO instruction will cause a jump to the destination given in the instruction, if the overflow flag is set.

- ADD AL, BL — Add signed bytes in AL and BL
- JO ERROR — Jump to label ERROR if overflow from add

JNO (JUMP IF NO OVERFLOW)

The overflow flag will be set if some signed arithmetic operation is too large to fit in the destination register or memory location. The JNO instruction will cause a jump to the destination given in the instruction, if the overflow flag is not set.

- ADD AL, BL — Add signed byte in AL and BL
  JNO DONE — Process DONE if no overflow

JCXZ (JUMP IF THE CX REGISTER IS ZERO)

This instruction will cause a jump to the label to a given in the instruction, if the CX register contains all 0's. The instruction does not look at the zero flag when it decides whether to jump or not.

- JCXZ SKIP — If CX = 0, skip the process
  SUB [BX], 07H — Subtract 7 from data value
  SKIP: ADD C — Next instruction

LOOP (JUMP TO SPECIFIED LABEL IF CX □ 0 AFTER AUTO DECREMENT)

This instruction is used to repeat a series of instructions some number of times. The number of times the instruction sequence is to be repeated is loaded into CX. Each time the LOOP instruction executes, CX is automatically decremented by 1. If CX is not 0, execution will jump to a destination specified by a label in the instruction. If CX = 0 after the auto decrement, execution will simply go on to the next instruction after LOOP. The destination address for the jump must be in the range of –128 bytes to +127 bytes from the address of the instruction after the LOOP instruction. This instruction does not affect any flag.

- MOV BX, OFFSET PRICES — Point BX at first element in array
  MOV CX, 40 — Load CX with number of elements in array
  NEXT: MOV AL, [BX] — Get element from array
  INC AL — Increment the content of AL
  MOV [BX], AL — Put result back in array
  INC BX — Increment BX to point to next location
  LOOP NEXT — Repeat until all elements adjusted

LOOPE / LOOPZ (LOOP WHILE CX □ 0 AND ZF = 1)

This instruction is used to repeat a group of instructions some number of times, or until the zero flag becomes 0. The number of times the instruction sequence is to be

repeated is loaded into CX. Each time the LOOP instruction executes, CX is automatically decremented by 1. If CX ≠0 and ZF = 1, execution will jump to a destination specified by a label in the instruction. If CX = 0, execution simply go on the next instruction after LOOPE / LOOPZ. In other words, the two ways to exit the loop are CX = 0 or ZF = 0. The destination address for the jump must be in the range of –128 bytes to +127 bytes from the address of the instruction after the LOOPE / LOOPZ instruction. This instruction does not affect any flag.

- MOV BX, OFFSET ARRAY Point BX to address of ARRAY
  before start of array DEC BX Decrement BX
  MOV CX, 100 Put number of array elements in CX
  NEXT: INC BX Point to next element in array
  CMP [BX], OFFH Compare array element with FFH LOOPE
  NEXT

LOOPNE / LOOPNZ (LOOP WHILE CX □ 0 AND ZF = 0)

This instruction is used to repeat a group of instructions some number of times, or until the zero flag becomes a 1. The number of times the instruction sequence is to be repeated is loaded into the count register CX. Each time the LOOPNE / LOOPNZ instruction executes, CX is automatically decremented by 1. If CX ▪ 0 and ZF = 0, execution will jump to a destination specified by a label in the instruction. If CX = 0, after the auto decrement or if ZF = 1, execution simply go on the next instruction after LOOPNE / LOOPNZ. In other words, the two ways to exit the loop are CX = 0 or ZF = 1. The destination address for the jump must be in the range of –128 bytes to +127 bytes from the address of the instruction after the LOOPNE / LOOPZ instruction. This instruction does not affect any flags.

- MOV BX, OFFSET ARRAY Point BX to adjust before start of array
  DEC BX Decrement BX
  MOV CX, 100 Put number of array in CX
  NEXT: INC BX Point to next element in array
  CMP [BX], ODH Compare array element with 0DH
  LOOPNZ NEXT

CALL (CALL A PROCEDURE)

The CALL instruction is used to transfer execution to a subprogram or a procedure. There two basic type of calls near and far. 1. A near call is a call to a procedure, which is in the same code segment as the CALL instruction. When the 8086 executes a near CALL instruction, it decrements the stack pointer by 2 and copies the offset of the next instruction after the CALL into the stack. This offset saved in the stack is referred to as the return address, because this is the address that execution will return to after the procedure is executed. A near CALL instruction will also load the instruction pointer with the offset of the first instruction in the procedure.

A RET instruction at the end of the procedure will return execution to the offset saved on the stack which is copied back to IP. 2. A far call is a call to a procedure, which is in a different segment from the one that contains the CALL instruction. When the 8086 executes a far call, it decrements the stack pointer by 2 and copies the content of the CS register to the stack. It then decrements the stack pointer by 2 again and copies the offset of the instruction after the CALL instruction to the stack. Finally, it loads CS with the segment base of the segment that contains the procedure, and loads IP with the offset of the first instruction of the procedure in that segment. A RET instruction at the end of the procedure will return execution to the next instruction after the CALL by restoring the saved values of CS and IP from the stack.

- CALL MULT
  This is a direct within segment (near or intra segment) call. MULT is the name of the procedure. The assembler determines the displacement of MULT from the instruction after the CALL and codes this displacement in as part of the instruction.

- CALL BX
  This is an indirect within-segment (near or intra-segment) call. BX contains the offset of the first instruction of the procedure. It replaces content of IP with content of register BX.

- CALL WORD PTR [BX]
  This is an indirect within-segment (near or intra-segment) call. Offset of the first instruction of the procedure is in two memory addresses in DS. Replaces content of IP with content of word memory location in DS pointed to by BX.

- CALL DIVIDE
  This is a direct call to another segment (far or inter-segment call). DIVIDE is the name of the procedure. The procedure must be declared far with DIVIDE PROC FAR at its start. The assembler will determine the code segment base for the segment that contains the procedure and the offset of the start of the procedure. It will put these values in as part of the instruction code.

- CALL DWORD PTR [BX]
  This is an indirect call to another segment (far or inter-segment call). New values for CS and IP are fetched from four-memory location in DS. The new value for CS is fetched from [BX] and [BX + 1]; the new IP is fetched from [BX + 2] and [BX +3].

RET (RETURN EXECUTION FROM PROCEDURE TO CALLING PROGRAM)

The RET instruction will return execution from a procedure to the next instruction after the CALL instruction which was used to call the procedure. If the procedure is near procedure (in the same code segment as the CALL instruction), then the return will be done by replacing the IP with a word from the top of the stack. The word from the top of the stack is the offset of the next instruction after the CALL. This offset was pushed into the stack as part of the operation of the CALL instruction. The stack pointer will be incremented by 2 after the return address is popped off the stack.

If the procedure is a far procedure (in a code segment other than the one from which it is called), then the instruction pointer will be replaced by the word at the top of the stack. This word is the offset part of the return address put there by the CALL instruction. The stack pointer will then be incremented by 2. The CS register is then replaced with a word from the new top of the stack. This word is the segment base part of the return address that was pushed onto the stack by a far call operation. After this, the stack pointer is again incremented by 2.

A RET instruction can be followed by a number, for example, RET 6. In this case, the stack pointer will be incremented by an additional six addresses after the IP when the IP and CS are popped off the stack. This form is used to increment the stack pointer over parameters passed to the procedure on the stack.

The RET instruction does not affect any flag.

VII. PROCESS CONTROL INSTRUCTIONS

- **STC (SET CARRY FLAG)**
  This instruction sets the carry flag to 1. It does not affect any other flag.
- **CLC (CLEAR CARRY FLAG)**
  This instruction resets the carry flag to 0. It does not affect any other flag.
- **CMC (COMPLEMENT CARRY FLAG)**
  This instruction complements the carry flag. It does not affect any other flag.
- **STD (SET DIRECTION FLAG)**
  This instruction sets the direction flag to 1. It does not affect any other flag.
- **CLD (CLEAR DIRECTION FLAG)**
  This instruction resets the direction flag to 0. It does not affect any other flag.
- **STI (SET INTERRUPT FLAG)**
  Setting the interrupt flag to a 1 enables the INTR interrupt input of the 8086. The instruction will not take affect until the next instruction after STI. When the INTR input is enabled, an interrupt signal on this input will then cause the 8086 to interrupt program execution, push the return address and flags on the stack, and execute an interrupt service procedure. An IRET instruction at the end of the interrupt service procedure will restore the return address and flags that were pushed onto the stack and return execution to the interrupted program. STI does not affect any other flag.
- **CLI (CLEAR INTERRUPT FLAG)**

This instruction resets the interrupt flag to 0. If the interrupt flag is reset, the 8086 will not respond to an interrupt signal on its INTR input. The CLI instructions, however, has no effect on the non-maskable interrupt input, NMI. It does not affect any other flag.

- **HLT (HALT PROCESSING)**
  The HLT instruction causes the 8086 to stop fetching and executing instructions. The 8086 will enter a halt state. The different ways to get the processor out of the halt state are with an interrupt signal on the INTR pin, an interrupt signal on the NMI pin, or a reset signal on the RESET input.
- **NOP (PERFORM NO OPERATION)**
  This instruction simply uses up three clock cycles and increments the instruction pointer to point to the next instruction. The NOP instruction can be used to increase the delay of a delay loop. When hand coding, a NOP can also be used to hold a place in a program for an instruction that will be added later. NOP does not affect any flag.
- **ESC (ESCAPE)**
  This instruction is used to pass instructions to a coprocessor, such as the 8087 Math coprocessor, which shares the address and data bus with 8086. Instructions for the coprocessor are represented by a 6-bit code embedded in the ESC instruction. As the 8086 fetches instruction bytes, the coprocessor also fetches these bytes from the data bus and puts them in its queue. However, the coprocessor treats all the normal 8086 instructions as NOPs. When 8086 fetches an ESC instruction, the coprocessor decodes the instruction and carries out the action specified by the 6-bit code specified in the instruction. In most cases, the 8086 treats the ESC instruction as a NOP. In some cases, the 8086 will access a data item in memory for the coprocessor.

PROCEDURE

Procedure is a part of code that can be called from your program in order to make some specific task. Procedures make program more structural and easier to understand. Generally procedure returns to the same point from where it was called. The syntax for procedure declaration:

- name PROC
  ; here goes the code
  ; of the procedure
  .
  .
  RET
  name ENDP

name - is the procedure name, the same name should be in the top and the bottom, this is used to check correct closing of procedures.

ASSEMBLER DIRECTIVES

- **PROC**
- **ENDP**

PROC and **ENDP** are compiler directives, so they are not assembled into any real machine code. Compiler just remembers the address of procedure. CALL instruction is used to call a procedure.

- **ORG**

Eg: ORG 100h

ORG 100h is a compiler directive (it says to compiler how to handle the source code). This directive is very important when you work with variables. It says to compiler that the executable file will be loaded at the offset of 100h (256 bytes), so compiler should calculate the correct address for all variables when it replaces the variable names with their offsets. Directives are never converted to any real machine code.

- **ENDS**

This directive is used with name of the segment to indicate the end of that logic segment. For example

```
CODE SEGMENT ; this statement starts the segment
.
.
CODE ENDS ; this statement ends the segment
```

- **ASSUME**

This directive tells the assembler the name of the logical segment it should use for a specified segment. For example ASSUME CS:CODE, tells the assembler that the instructions for a program are in a logical segment named CODE. The 8086 works directly with only 4 physical segments: a Code segment, a data segment, a stack segment, and an extra segment.

8086 PROGRAMS

Program for find the ADDITION of two numbers:	**Program for find the SUBTRACTION of two numbers:**
MOV AX,05	MOV AX,05
MOV BX,03	MOV BX,03

ADD AX,BX
MOV SI,8000
MOV [SI],AX
INT 03
	SUB AX,BX
	MOV SI,8000
	MOV [SI],AX
	INT 03

Program for find the MULTIPLICATION of two numbers:
MOV AX,05
MOV BX,03
MUL BX
MOV SI,8000
MOV [SI],AX
INT 03
	Program for find the DIVISION of two numbers:
	MOV AX,05
	MOV BX,03
	DIV BX
	MOV SI,8000
	MOV [SI],AX
	INT 03

Program for find the GREATEST OF 2-NUMBERS:
MOV AX,04
MOV BX,05
CMP AX,BX
JNC LABEL1
MOV SI,8001
MOV [SI],AX
LABEL1: MOV [SI],BX
INT 03
	Program for find the AVERAGE OF N-NUMBERS:
	MOV AX,0000
	MOV SI,8000
	MOV DI,8020
	MOV CX,5
	LABLE1: ADD AX,[SI]
	INC SI
	INC SI
	LOOP LABLE1
	DIV CX
	MOV [DI],AX
	INT 03

Program for find the SUM OF N-NUMBERS:	**Program for find the find the factorial of a number**

MOV SI,8000
MOV CX,[SI]
MOV AX,0000
MOV BX,ax
LABEL1: INC BX
ADD AX,BX
CMP BX,CX
JNZ LABEL1
MOV DI,8010
MOV [DI],AX
INT 03
	MOV SI,8000
	MOV BX[SI]
	MOV AX,01
	LABEL1: MUL BX
	DEC BX
	JNZ LABEL1
	MOV DI,8050
	MOV [DI],AX
	INT 03

DIFFERENCE BETWEEN MIMIMUM MODE AND MAXIMUM MODE

Minimum mode	Maximum mode
In minimum mode there can be only one processor i.e. 8086.	In maximum mode there can be multiple processors with 8086, like 8087 and 8089.
$MN/\overline{MX}$ is 1 to indicate minimum mode.	$MN/\overline{MX}$ is 0 to indicate maximum mode.
ALE for the latch is given by 8086 as it is the only processor in the circuit.	ALE for the latch is given by 8288 bus controller as there can be multiple processors in the circuit.
$\overline{DEN}$ and $DT/\overline{R}$ for the trans-receivers are given by 8086 itself.	and $DT/\overline{R}$ for the trans-receivers are given by 8288 bus controller.
Direct control signals $M/\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ are given by 8086.	Instead of control signals, each processor generates status signals called $\overline{S_2}$, $\overline{S_1}$ and $\overline{S_0}$.
Control signals $M/\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ are decoded by a 3:8 decoder like 74138.	Status signals $\overline{S_2}$, $\overline{S_1}$ and $\overline{S_0}$ are decoded by a bus controller like 8288 to produce control signals.
$\overline{INTA}$ is given by 8086 in response to an interrupt on INTR line.	$\overline{INTA}$ is given by 8288 bus controller in response to an interrupt on INTR line.
HOLD and HLDA signals are used for bus request with a DMA controller like 8237.	$\overline{RQ}/\overline{GT}$,lines are used for bus requests by other processors like 8087 or 8089.
The circuit is simpler.	The circuit is more complex.
Multiprocessing cannot be performed hence performance is lower.	As multiprocessing can be performed, it can give very high performance.

Minimum Mode Configuration For 8086

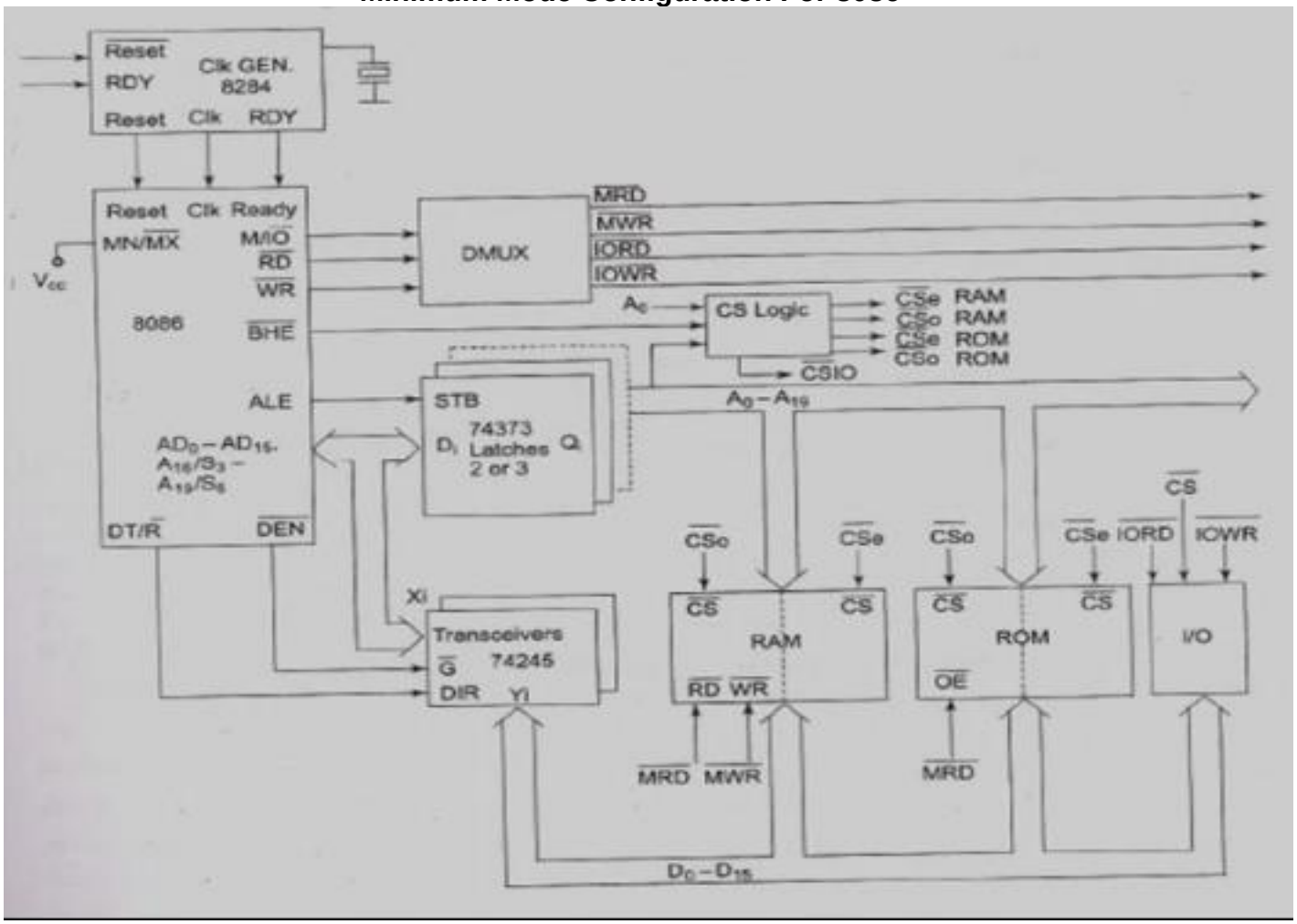


Fig 2.5 : Minimum Mode Configuration For 8086

Maximum Mode Configuration For 8086

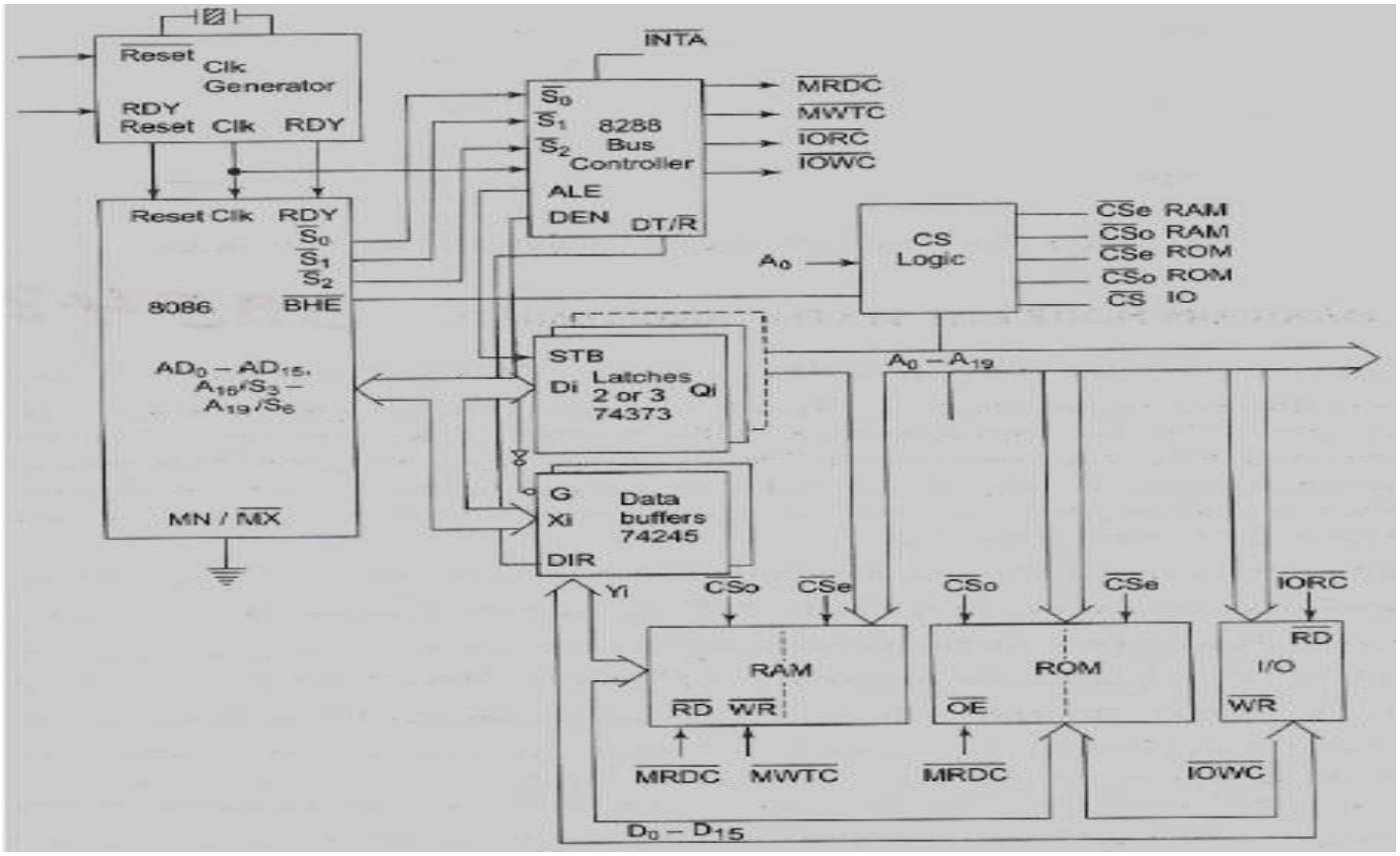


Fig 2.6 : Maximum Mode Configuration For 8086

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

SCHOOL OF ELECTRICAL AND ELECTRONICS
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

UNIT 3
Microprocessor , Interfacing and Its Applications-SECA1508

8255 - PROGRAMMABLE PERIPHERAL INTERFACE (PPI)

The **Intel 8255** (or **i8255**) Programmable PeripheralInterface (**PPI**) chip is a peripheral chip, is used to give the CPU access to programmable parallel I/O. It can be programmable to transfer data under various conditions from simple I/O to interrupt I/O. it is flexible versatile and economical (when multiple I/O ports are required) but somewhat complex. It is an important general purpose I/O device that can be used with almost any microprocessor.

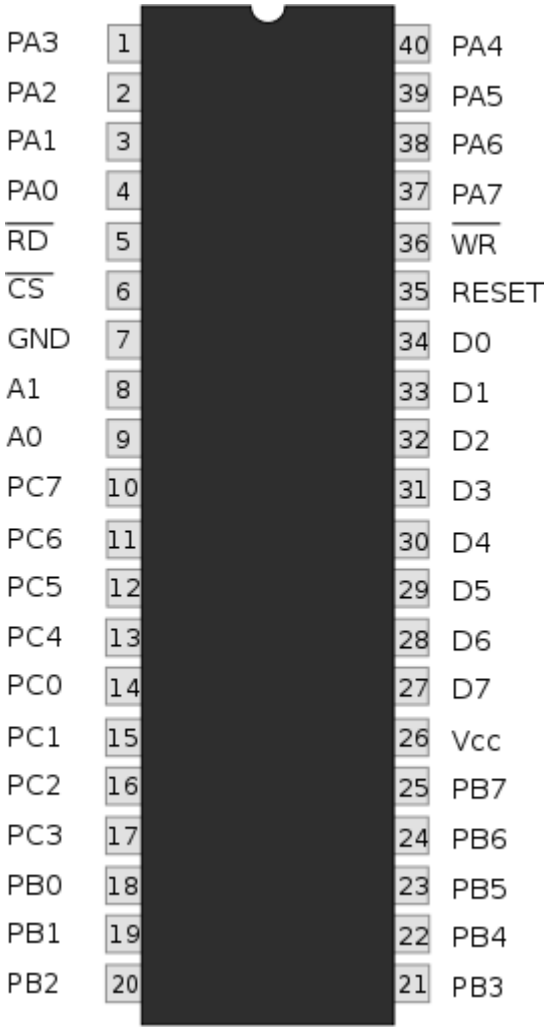


Fig 3.1: Pin diagram of 8255

I. Functional block of 8255 – Programmable Peripheral Interface (PPI)

The 8255A has 24 I/O pins that can be grouped primarily in two 8-bit parallel ports: A and B with the remaining eight bits as port C. The eight bits of port C can be used as individual bits or be grouped in to 4-bit ports: CUpper (Cu) and CLower (CL) as in Figure 2. The function of these ports is defined by writing a control word in the control register as shown in Figure

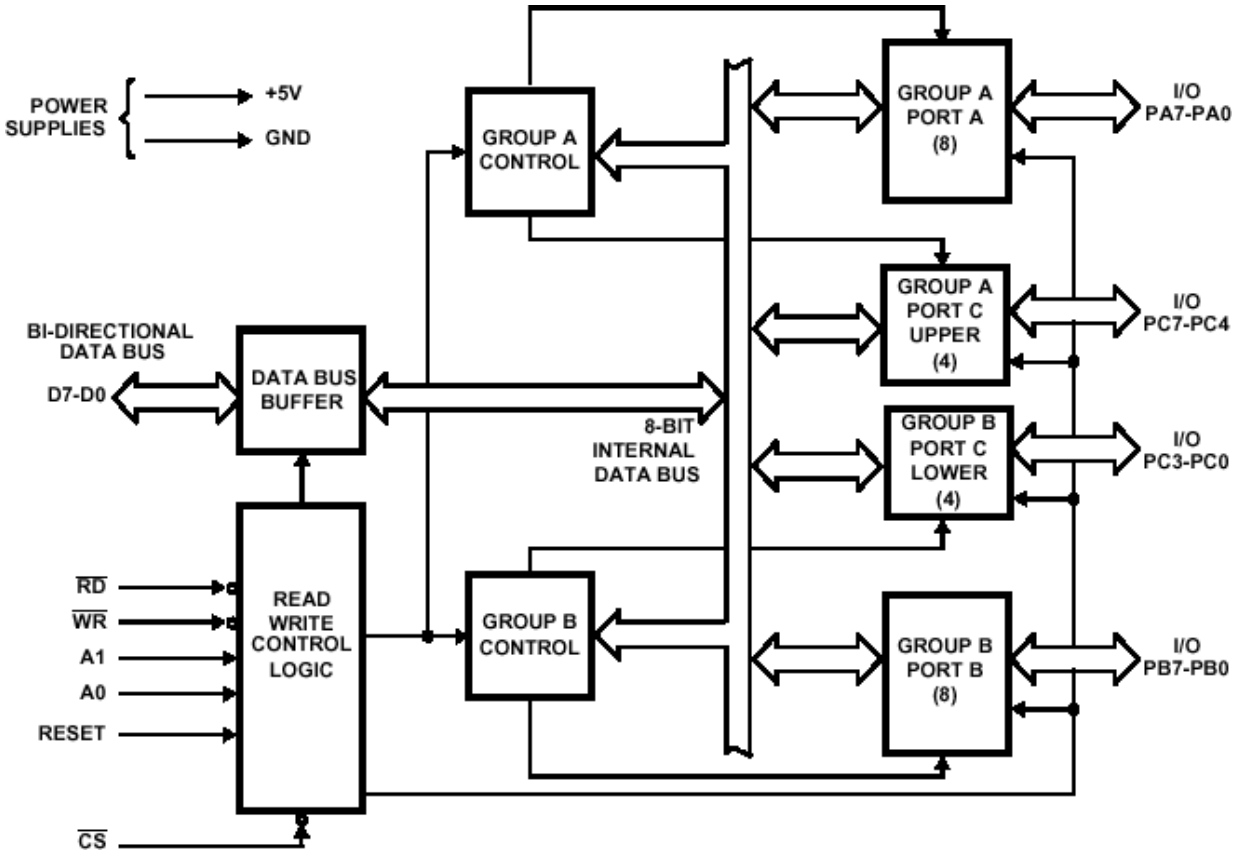


Fig 3. 2. Block diagram of 8255

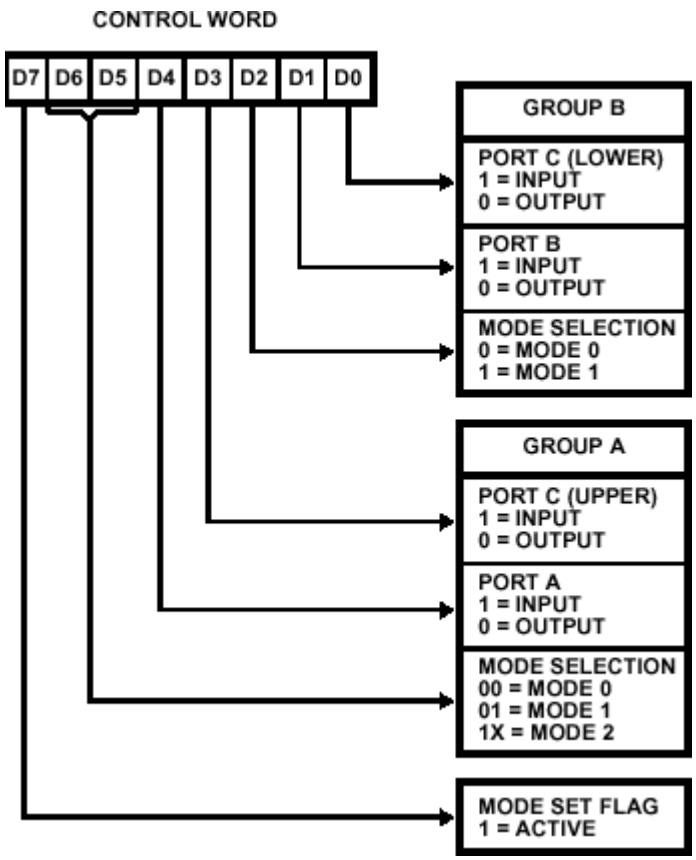


Fig 3.3. Control word Register format

Data Bus Buffer

This three-state bi-directional 8-bit buffer is used to interface the 8255 to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

(CS) Chip Select. A "low" on this input pin enables the communication between the 8255 and the CPU.

(RD) Read. A "low" on this input pin enables 8255 to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 8255.

(WR) Write. A "low" on this input pin enables the CPU to write data or control words into the 8255.

(A0 and A1) Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word register. They are normally connected to the least significant bits of the address bus (A0 and A1).

(RESET) Reset. A "high" on this input initializes the control register to 9Bh and all ports (A, B, C) are set to the input mode.

A1	A0	SELECTION
0	0	PORT A
0	1	PORT B
1	0	PORT C
1	1	CONTROL

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 8255. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255. Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Ports A, B, and C

The 8255 contains three 8-bit ports (A, B, and C). All can be configured to a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255.

Port A One 8-bit data output latch/buffer and one 8-bit data input latch. Both "pull-up" and "pull-down" bus-hold devices are present on Port A.

Port B One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal output and status signal inputs in conjunction with ports A and B.

II. Operational modes of 8255

There are two basic operational modes of 8255:

1. Bit set/reset Mode (BSR Mode).
2. Input/Output Mode (I/O Mode).

The two modes are selected on the basis of the value present at the D_7 bit of the Control Word Register. When D_7 = 1, 8255 operates in I/O mode and when D_7 = 0, it operates in the BSR mode.

1. Bit set/reset (BSR) mode

The Bit Set/Reset (BSR) mode is applicable to port C only. Each line of port C (PC_0 - PC_7) can be set/reset by suitably loading the control word register as shown in Figure 4. BSR mode and I/O mode are independent and selection of BSR mode does not affect the operation of other ports in I/O mode.

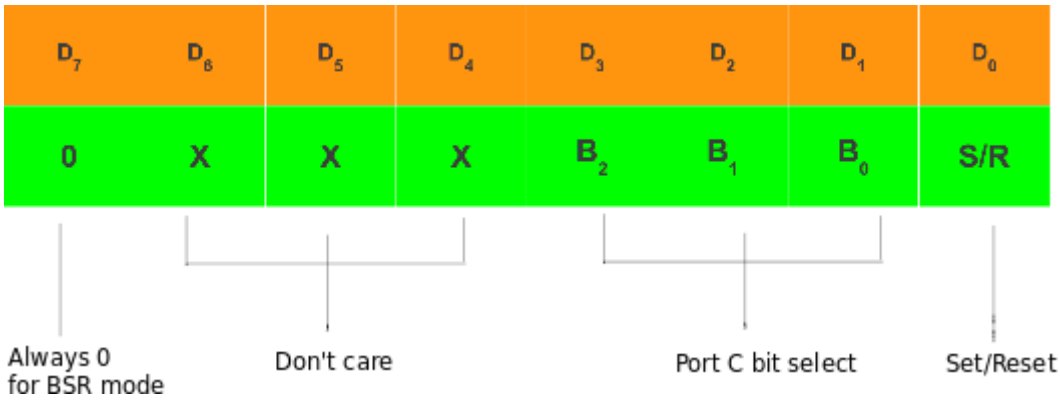


Fig 3.4: 8255 Control register format for BSR mode

- D_7 bit is always 0 for BSR mode.
- Bits D_6, D_5 and D_4 are don't care bits.
- Bits D_3, D_2 and D_1 are used to select the pin of Port C.
- Bit D_0 is used to set/reset the selected pin of Port C.

Selection of port C pin is determined as follows:

B3	B2	B1	Bit/pin of port C selected
0	0	0	PC_0
0	0	1	PC_1
0	1	0	PC_2
0	1	1	PC_3
1	0	0	PC_4
1	0	1	PC_5
1	1	0	PC_6
1	1	1	PC_7

As an example, if it is needed that PC_5 be set, then in the control word,

1. Since it is BSR mode, **D_7 = '0'**.
2. Since D_4, D_5, D_6 are not used, assume them to be '**0**'.
3. PC_5 has to be selected, hence, **D_3 = '1', D_2 = '0', D_1 = '1'**.
4. PC_5 has to be set, hence, **D0 = '1'**.

Thus, as per the above values, 0B (Hex) will be loaded into the Control Word Register (CWR).

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	0	1	1

2. Input/Output mode

This mode is selected when D_7 bit of the Control Word Register is 1. There are three I/O modes:

1. Mode 0 - Simple I/O
2. Mode 1 - Strobed I/O
3. Mode 2 - Strobed Bi-directional I/O

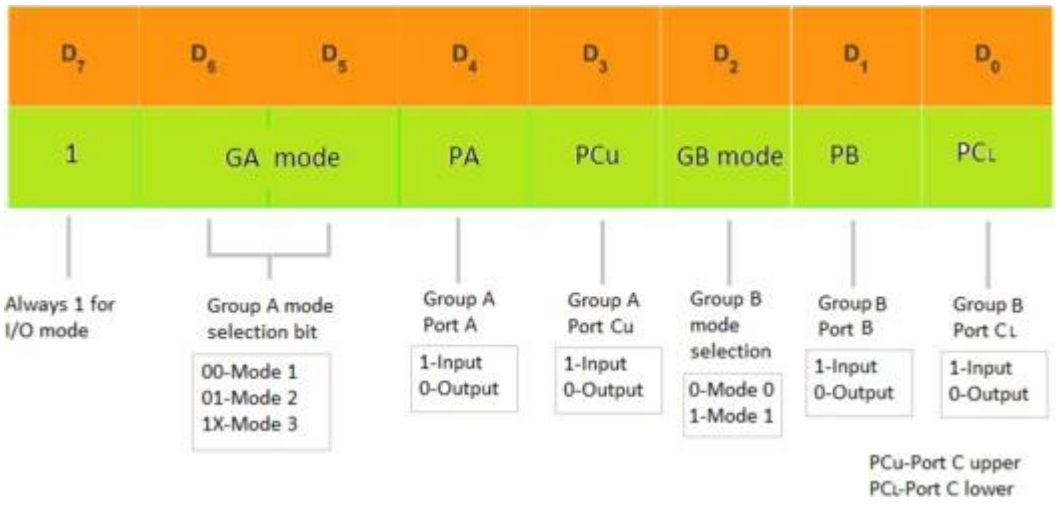


Figure 3.5: 8255 Control word for I/O mode

- **D_0, D_1, D_3, D_4** are assigned for lower port C, port B, upper port C and port A respectively. When these bits are **1**, the corresponding port acts as an input port. For e.g., if $D_0 = D_4 = 1$, then lower port C and port A act as input ports. If these bits are **0**, then the corresponding port acts as an output port. For e.g., if $D_1 = D_3 = 0$, then port B and upper port C act as output ports as shown in Figure 5.
- **D_2** is used for mode selection of Group B (port B and lower port C). When $D_2 = 0$, mode 0 is selected and when $D_2 = 1$, mode 1 is selected.
- **D_5& D_6** are used for mode selection of Group A (port A and upper port C). The selection is done as follows:

D_6	D_5	Mode
0	0	0
0	1	1
1	X	2

- As it is I/O mode, **D_7** = 1.

For example, if port B and upper port C have to be initialized as input ports and lower port C and port A as output ports (all in mode 0):

1. Since it is an I/O mode, $D_7 = 1$.
2. Mode selection bits, D2, D5, D6 are all 0 for mode 0 operation.

3. Port B and upper port C should operate as Input ports, hence, $D_1 = D_3 = 1$.
4. Port A and lower port C should operate as Output ports, hence, $D_4 = D_0 = 0$.

Hence, for the desired operation, the control word register will have to be loaded with **"10001010" = 8A (hex)**.

- **Mode 0 - simple I/O**

In this mode, the ports can be used for simple I/O operations without handshaking signals. Port A, port B provide simple I/O operation. 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. Output ports are latched.
2. Input ports are buffered, not latched.
3. Ports do not have handshake or interrupt capability.
4. With 4 ports, 16 different combinations of I/O are possible.

➢ Mode 0 – input mode

- In the input mode, the 8255 gets data from the external peripheral ports and the CPU reads the received data via its data bus.
- The CPU first selects the 8255 chip by making ¬CS low. Then it selects the desired port using A_0 and A_1 lines.
- The CPU then issues an ¬RD signal to read the data from the external peripheral device via the system data bus.

➢ Mode 0 - output mode

- In the output mode, the CPU sends data to 8255 via system data bus and then the external peripheral ports receive this data via 8255 port.
- CPU first selects the 8255 chip by making ¬CS low. It then selects the desired port using A_0 and A_1 lines.
- CPU then issues a ¬WR signal to write data to the selected port via the system data bus. This data is then received by the external peripheral device connected to the selected port.

➢ 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, i.e., for e.g., 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 used as 8-bit i/o ports.
2. Each port uses three lines of port c as handshake signal and remaining two signals can be used as i/o ports.
3. Interrupt logic is supported.
4. Input and Output data are latched.

Input Handshaking signals

1. IBF (Input Buffer Full) - It is an output indicating that the input latch contains information.

2. STB (Strobed Input) - The strobe input loads data into the port latch, which holds the information until it is input to the microprocessor via the IN instruction.

3. INTR (Interrupt request) - It is an output that requests an interrupt. The INTR pin becomes a logic 1 when the STB input returns to a logic 1, and is cleared when the data are input from the port by the microprocessor.

4. INTE (Interrupt enable) - It is neither an input nor an output; it is an internal bit programmed via the port PC4(port A) or PC2(port B) bit position.

Output Handshaking signals

1. OBF (Output Buffer Full) - It is an output that goes low whenever data are output(OUT) to the port A or port B latch. This signal is set to a logic 1 whenever the ACK pulse returns from the external device.

2. ACK (Acknowledge)-It causes the OBF pin to return to a logic 1 level. The ACK signal is a response from an external device, indicating that it has received the data from the 82C55 port.

3. INTR (Interrupt request) - It is a signal that often interrupts the microprocessor when the external device receives the data via the signal. this pin is qualified by the internal INTE(interrupt enable) bit.

4. INTE (Interrupt enable) - It is neither an input nor an output; it is an internal bit programmed to enable or disable the INTR pin. The INTE A bit is programmed using the PC6 bit and INTE B is programmed using the PC2 bit.

- **Mode 2**

Only group A can be initialized 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 initialized in mode 0 or as handshaking for port B if group B is initialized in mode 1. 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. Acknowledgement and handshaking signals are provided to maintain proper data flow and synchronisation between the data transmitter and receiver.

III. Interfacing 8255 with 8085 processor

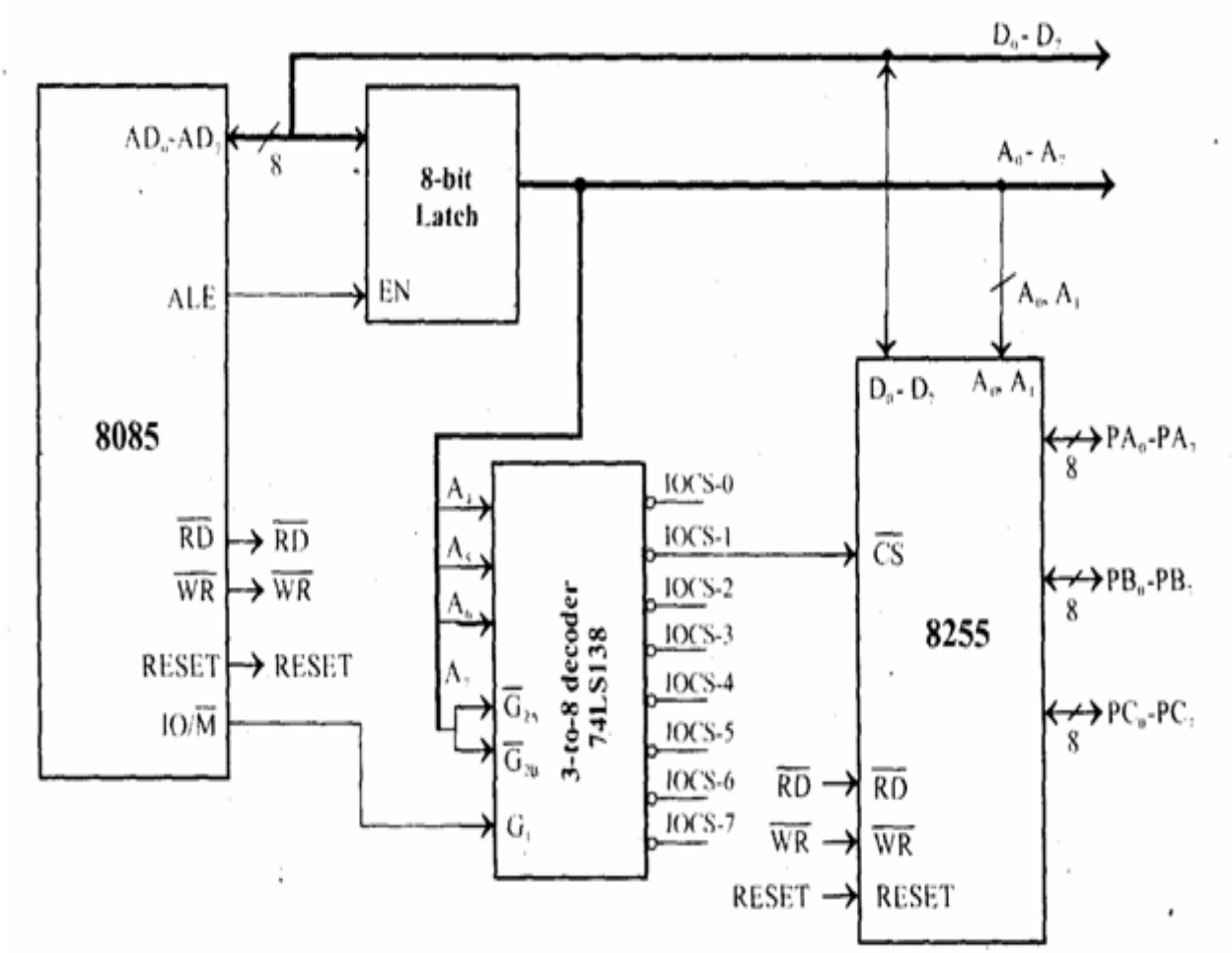


Fig 3.6. Interfacing 8255 with 8085 processor

- The 8255 can be either memory mapped or I/O mapped in the system. In the schematic shown in above is I/O mapped in the system.
- Using a 3-to-8 decoder generates the chip select signals for I/O mapped devices.
- The address lines A4, A5 and A6 are decoded to generate eight chip select signals (IOCS-0 to IOCS-7) and in this, the chip select IOCS- 1 is used to select 8255 as shown in Figure 6.
- The address line A7 and the control signal IO/M (low) are used as enable for the decoder.

- The address line A0 of 8085 is connected to A0 of 8255 and A1 of 8085 is connected to A1 of 8255 to provide the internal addresses.
- The data lines D0-D7 are connected to D0-D7 of the processor to achieve parallel data transfer.
- The I/O addresses allotted to the internal devices of 8255 are listed in table.

Internal Device	Binary Address								Hexa Address
	Decoder input and enable				Input to address pins of 8255				
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Port-A	0	0	0	1	x	x	0	0	10
Port-B	0	0	0	1	x	x	0	1	11
Port-C	0	0	0	1	x	x	1	0	12
Control Register	0	0	0	1	x	x	1	1	13

Note : *Don't care "x" is considered as zero.*

USART 8251 (Universal Synchronous/ Asynchronous Receiver Transmitter)

The 8251 is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication. As a peripheral device of a microcomputer system, the 8251 receives 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 as shown in Figure 10.

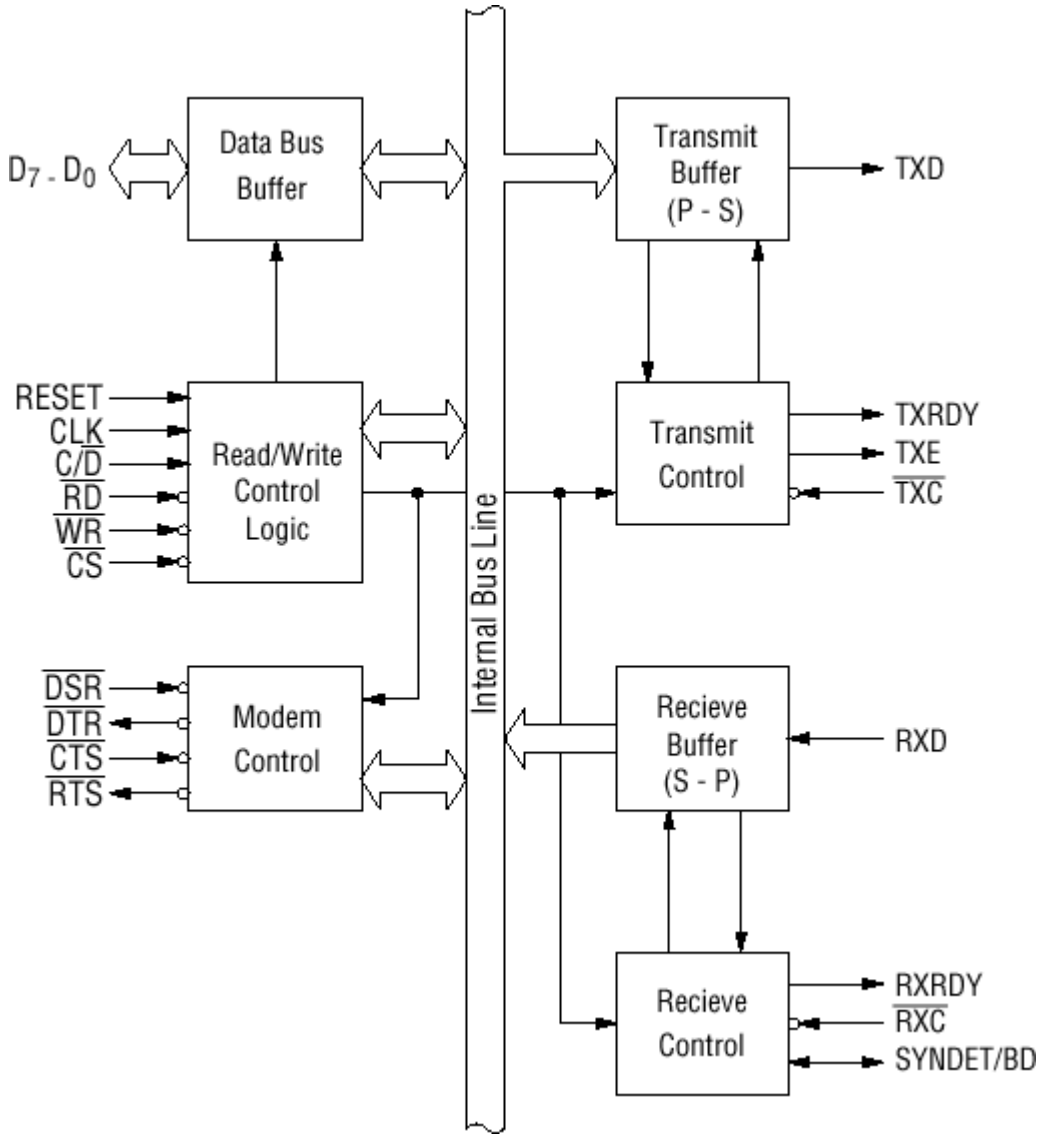


Figure 3.7 : Architecture of 8251

Transmitter Section

The transmitter section consists of three blocks–transmitter buffer register, output register and the transmitter control logic block. The CPU deposits (when TXRDY = 1, meaning that the transmitter buffer register is empty) data into the transmitter buffer register, which is subsequently put into the output register (when TXE = 1, meaning that the output buffer is empty). In the output register, the eight bit data is converted into serial form and comes out via TXD pin. The serial data bits are preceded by START bit and succeeded by STOP bit, which are known as framing bits. But this happens only if transmitter is enabled and the CTS is low. TXC signal is the transmitter clock signal which controls the bit rate on the TXD line (output line). This clock frequency can be 1, 16 or 64 times the baud.

Receiver Section

The receiver section consists of three blocks — receiver buffer register, input register and the receiver control logic block. Serial data from outside world is delivered to the input register via RXD line, which is subsequently put into parallel form and placed in the receiver buffer register. When this register is full, the RXRDY (receiver ready) line becomes high. This line is then used either to interrupt the MPU or to indicate its own status. MPU then accepts the data from the register. RXC line stands for receiver clock. This clock signal controls the rate at which bits are received by the input register. The clock can be set to 1, 16 or 64 times the baud in the asynchronous mode.

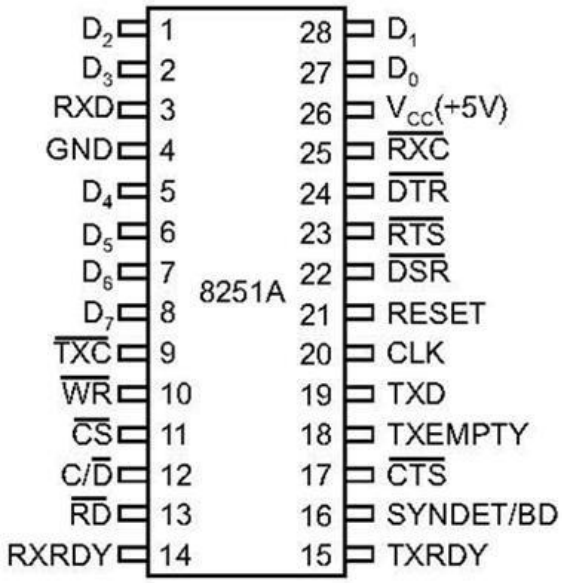


Fig 3.8 Pin Configuration of 8251

Pin Configuration of 8251 is shown in figure 11.

D 0 to D 7 (I/O terminal)

This is bidirectional data bus which receive control words and transmits data from the CPU and sends status words and received data to CPU.

RESET (Input terminal)

A "High" on this input forces the 8251 into "reset status." The device waits for the writing of "mode instruction." The min. reset width is six clock inputs during the operating status of CLK.

CLK (Input terminal)

CLK signal is used to generate internal device timing. CLK signal is independent of RXC or TXC. However, the frequency of CLK must be greater than 30 times the RXC and TXC at Synchronous mode and Asynchronous "x1" mode, and must be greater than 5 times at Asynchronous "x16" and "x64" mode.

WR (Input terminal)

This is the "active low" input terminal which receives a signal for writing transmit data and control words from the CPU into the 8251.

RD (Input terminal)

This is the "active low" input terminal which receives a signal for reading receive data and status words from the 8251.

C/D (Input terminal)

This is an input terminal which receives a signal for selecting data or command words and status words when the 8251 is accessed by the CPU. If C/D = low, data will be accessed. If C/D = high, command word or status word will be accessed.

CS (Input terminal)

This is the "active low" input terminal which selects the 8251 at low level when the CPU accesses. Note: The device won't be in "standby status"; only setting CS = High.

TXD (output terminal)

This is an output terminal for transmitting data from which serial-converted data is sent out. The device is in "mark status" (high level) after resetting or during a status when transmit is disabled. It is also possible to set the device in "break status" (low level) by a command.

TXRDY (output terminal)

This is an output terminal which indicates that the 8251is ready to accept a transmitted data character. But the terminal is always at low level if CTS = high or the device was set in "TX disable status" by a command. Note: TXRDY status word indicates that transmit data character is receivable, regardless of CTS or command. If the CPU writes a data character, TXRDY will be reset by the leading edge or WR signal.

TXEMPTY (Output terminal)

This is an output terminal which indicates that the 8251 has transmitted all the characters and had no data character. In "synchronous mode," the terminal is at high level, if transmit data characters are no longer remaining and sync characters are automatically transmitted. If the CPU writes a data character, TXEMPTY will be reset by the leading edge of WR signal. Note : As the transmitter is disabled by setting CTS "High" or command, data written before disable will be sent out. Then TXD and TXEMPTY will be "High". Even if a data is written after disable, that data is not sent out and TXE will be "High".After the transmitter is enabled, it sent out. (Refer to Timing Chart of Transmitter Control and Flag Timing)

TXC (Input terminal)

This is a clock input signal which determines the transfer speed of transmitted data. In "synchronous mode," the baud rate will be the same as the frequency of TXC. In "asynchronous mode", it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16 or 1/64 the TXC. The falling edge of TXC sifts the serial data out of the 8251.

RXD (input terminal)

This is a terminal which receives serial data.

RXRDY (Output terminal)

This is a terminal which indicates that the 8251 contains a character that is ready to READ. If the CPU reads a data character, RXRDY will be reset by the leading edge of RD signal. Unless the CPU reads a data character before the next one is received completely, the preceding data will be lost. In such a case, an overrun error flag status word will be set.

RXC (Input terminal)

This is a clock input signal which determines the transfer speed of received data. In "synchronous mode," the baud rate is the same as the frequency of RXC. In "asynchronous mode," it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16, 1/64 the RXC.

SYNDET/BD (Input or output terminal)

This is a terminal whose function changes according to mode. In "internal synchronous mode." this terminal is at high level, if sync characters are received and synchronized. If a status word is read, the terminal will be reset. In "external synchronous mode, "this is an input terminal. A "High" on this input forces the 8251 to start receiving data characters.

In "asynchronous mode," this is an output terminal which generates "high level"output upon the detection of a "break" character if receiver data contains a "low-level" space between the stop bits of two continuous characters. The terminal will be reset, if RXD is at high level. After Reset is active, the terminal will be output at low level.

DSR (Input terminal)

This is an input port for MODEM interface. The input status of the terminal can be recognized by the CPU reading status words.

DTR (Output terminal)

This is an output port for MODEM interface. It is possible to set the status of DTR by a command.

CTS (Input terminal)

This is an input terminal for MODEM interface which is used for controlling a transmit circuit. The terminal controls data transmission if the device is set in "TX Enable" status by a command. Data is transmitable if the terminal is at low level.

RTS (Output terminal)

This is an output port for MODEM interface. It is possible to set the status RTS by a command.

The 8251 functional configuration is programmed by software. Operation between the 8251 and a CPU is executed by program control. Table 1 shows the operation between a CPU and the device.

Summary of Control Signals for 8251

$\overline{CS}$	$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	*Function*
0	1	1	0	MPU writes instructions in the control register
0	1	0	1	MPU reads status from the status register
0	0	1	0	MPU outputs data to the Data Buffer
0	0	0	1	MPU accepts data from the Data Buffer
1	X	X	X	USART is not selected

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 12 and 13. 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.

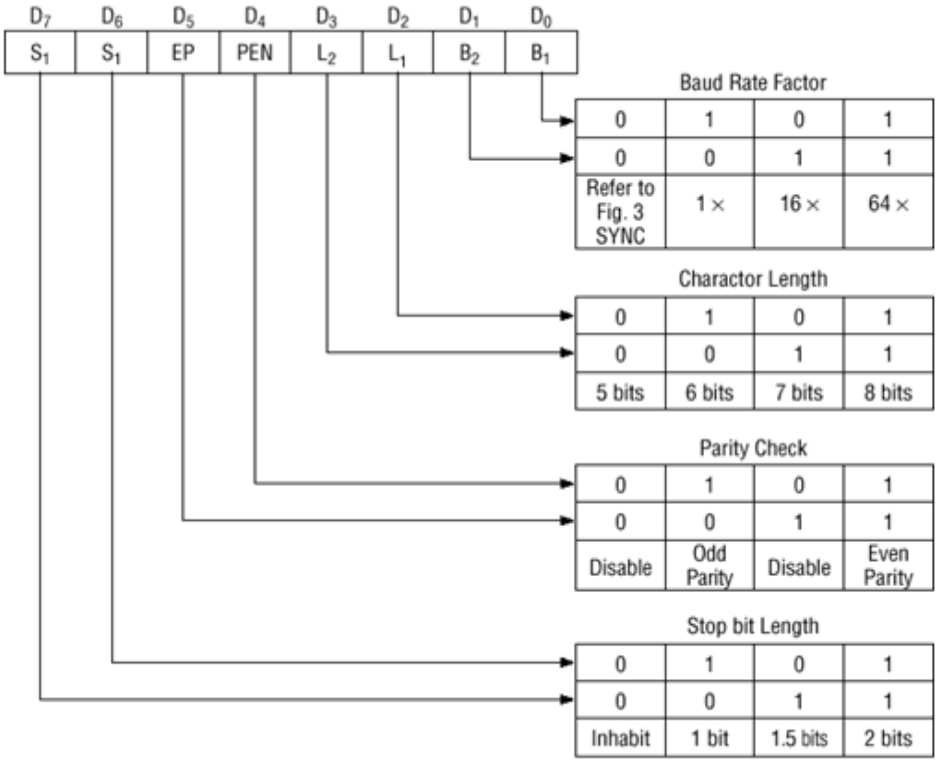


Fig 3.9: Bit configuration of mode instruction(asynchronous)

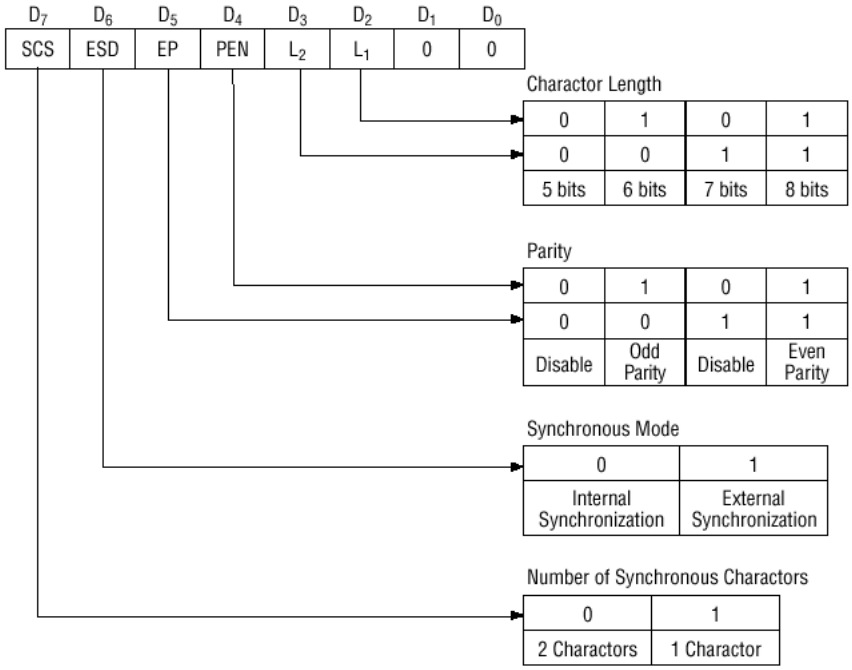


Fig 3.10 Bit configuration of mode instruction(synchronous)

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 as shown in figure 14.

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)

Status Word

It is possible to see the internal status of the 8251 by reading a status word. The bit configuration of status word is shown in Fig.15.

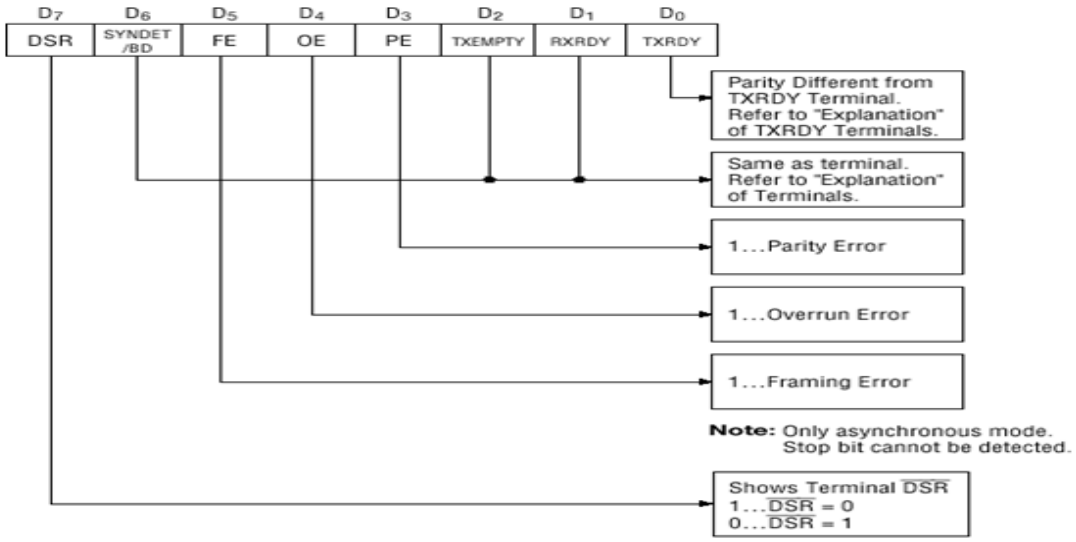


Fig 3.11: Bit configuration of Status Word

8253(8254) PROGRAMMABLE INTERVAL TIMER:

The 8254 programmable Interval timer consists of three independent 16-bit programmable counters (timers). Each counter is capable of counting in binary or binary coded decimal. The maximum allowable frequency to any counter is 10MHz. This device is useful whenever the microprocessor must control real-time events. The timer in a personal computer is an 8253. To operate a counter a 16-bit count is loaded in its register and on command, it begins to decrement the count until it reaches 0. At the end of the count it generates a pulse, which interrupts the processor. The count can count either in binary or BCD Each counter in the block diagram has 3 logical lines connected to it. Two of these lines, clock and gate, are inputs. The third, labeled OUT is an output.

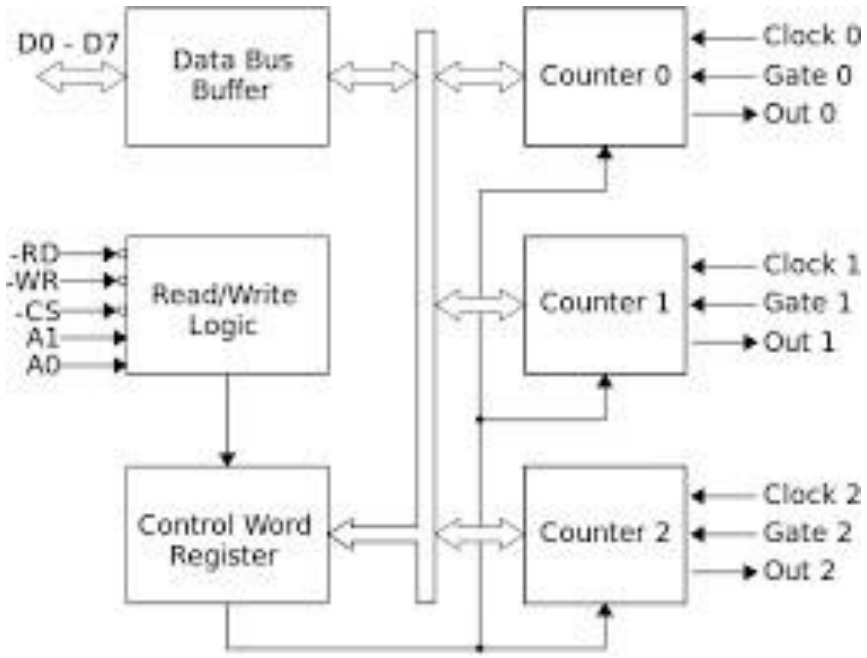


Fig : 3.12 Block Diagram of 8253 programmable interval timer

Data bus buffer- It is a communication path between the timer and the microprocessor. The buffer is 8-bit and bidirectional. It is connected to the data bus of the microprocessor. Read /write logic controls the reading and the writing of the counter registers. Control word register, specifies the counter to be used and either a Read or a write operation. Data is transmitted or received by the buffer upon execution of INPUT instruction from CPU as shown in figure 16. The data bus buffer has three basic functions,

(i). Programming the modes of 8253.

(ii). Loading the count value in times

(iii).Reading the count value from timers.

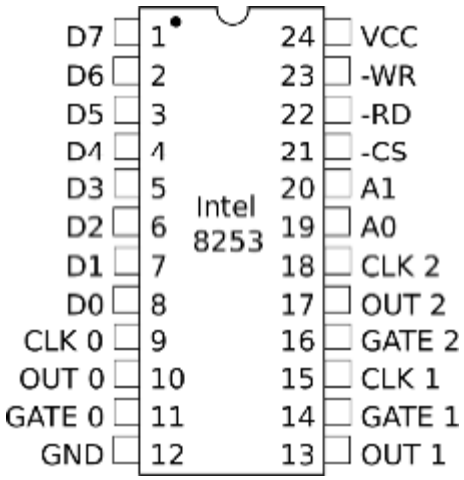


Fig : 3.13 Pin Diagram of 8253

The data bus buffer is connected to microprocessor using D7 – D0 pins which are also bidirectional. The data transfer is through these pins. These pins will be in high-impedance (or this state) condition until the 8253 is selected by a LOW or $\overline{CS}$ and either the read operation requested by a LOW $\overline{RD}$ on the input or a write operation $\overline{WR}$ requested by the input going LOW.

Read/ Write Logic:

It accepts inputs for the system control bus and in turn generation the control signals for overall device operation. It is enabled or disabled by $\overline{CS}$ so that no operation can occur to change the function unless the device has been selected as the system logic.

$\overline{CS}$:

The chip select input is used to enable the communicate between 8253 and the microprocessor by means of data bus. A low an $\overline{CS}$ enables the data bus buffers, while a high disables the buffer. The $\overline{CS}$ input does not have any affect on the operation of three times once they have been initialized. The normal configuration of a system employs an decode logic which actives $\overline{CS}$ line, whenever a specific set of addresses that correspond to 8253 appear on the address bus.

$\overline{RD}$ & $\overline{WR}$:

The read (RD) and write WR pins central the direction of data transfer on the 8-bit bus. When the input RD pin is low. Then CPU is inputting data from 8253 in the form of

counter value. When $\overline{WR}$ pins is low, then CPU is sending data to 8253 in the form of mode information or loading counters. The $\overline{RD}$ & $\overline{WR}$ should not both be low simultaneously. When $\overline{RD}$ & $\overline{WR}$ pins are HIGH, the data bus buffer is disabled.

A0 & A1:

These two input lines allow the microprocessor to specify which one of the internal register in the 8253 is going to be used for the data transfer. **Fig** shows how these two lines are used to select either the control word register or one of the 16-bit counters.

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	operation
0	1	0	0	0	Load counter '0'
0	1	0	0	1	Load counter '1'
0	1	0	1	0	Load counter '2'
0	1	0	1	1	Write mode word
0	0	1	0	0	Read TM_0
0	0	1	0	1	Read TM_1
0	0	1	1	0	Read TM_2
0	0	1	1	1	No- operation 3- state
1	X	X	X	X	Disable -- state
0	1	1	X	X	No- operation 3- state

Control word register:

It is selected when A0 and A1 . It the accepts information from the data bus buffer and stores it in a register. The information stored in then register controls the operation mode of each counter, selection of binary or BCD counting and the loading of each counting and the loading of each count register. This register can be written into, no read operation of this content is available.

Counters:

Each of the times has three pins associated with it. These are CLK (CLK) the gate (GATE) and the output (OUT).

CLK:

This clock input pin provides 16-bit times with the signal to causes the times to decrement max^{m} clock input is 2.6MHz. Note that the counters operate at the negative edge (H1 to L0) of this clock input. If the signal on this pin is generated by a fixed oscillator then the user has implemented a standard timer. If the input signal is a string of randomly occurring pulses, then it is called implementation of a counter.

GATE:

The gate input pin is used to initiate or enable counting. The exact

effect of the gate signal depends on which of the six modes of

operation is chosen.

OUTPUT:

The output pin provides an output from the timer. It actual use depends on the mode of operation of the timer. The counter can be read ¯in the fly‖ without inhibiting gate pulse or clock input.

CONTROL WORD OF 8253

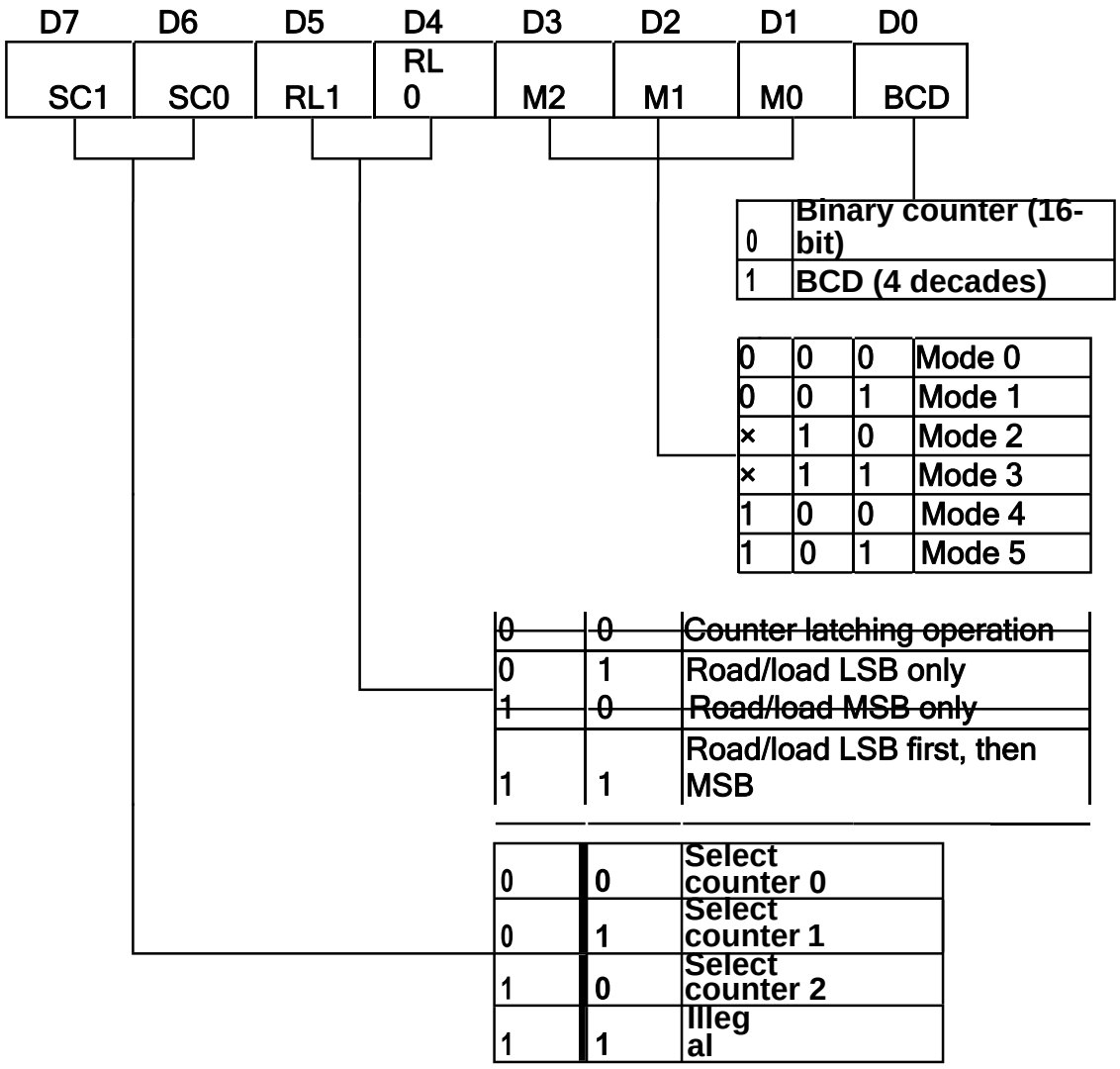


Fig : 3.14 Control Register

MODES OF OPERATION

Mode 0 Interrupt on terminal count
Mode 1 Programmable one shot
Mode 2 Rate Generator
Mode 3 Square wave rate Generator
Mode 4 Software triggered strobe
Mode 5 Hardware triggered strobe

Mode 0: The output goes high after the terminal count is reached. The counter stops if the Gate is low.. The timer count register is loaded with a count (say 6) when the WR line is made low by the processor. The counter unit starts counting down with each clock pulse. The output goes high when the register value reaches zero. In the mean time if the GATE is made low the count is suspended at the value(3) till the GATE is enabled again .

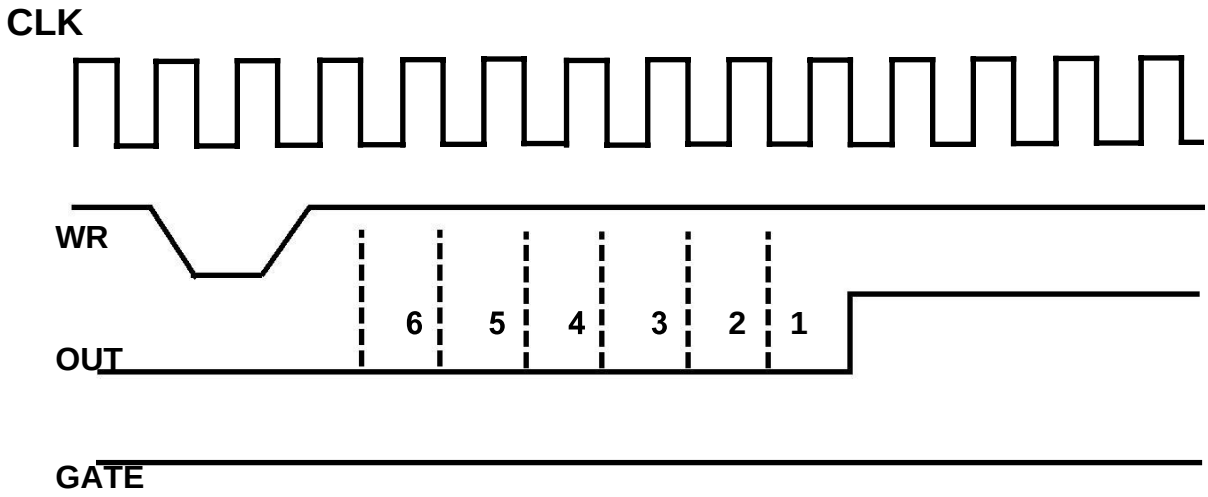


Fig 3.15 Mode 0 count when Gate is high (enabled)

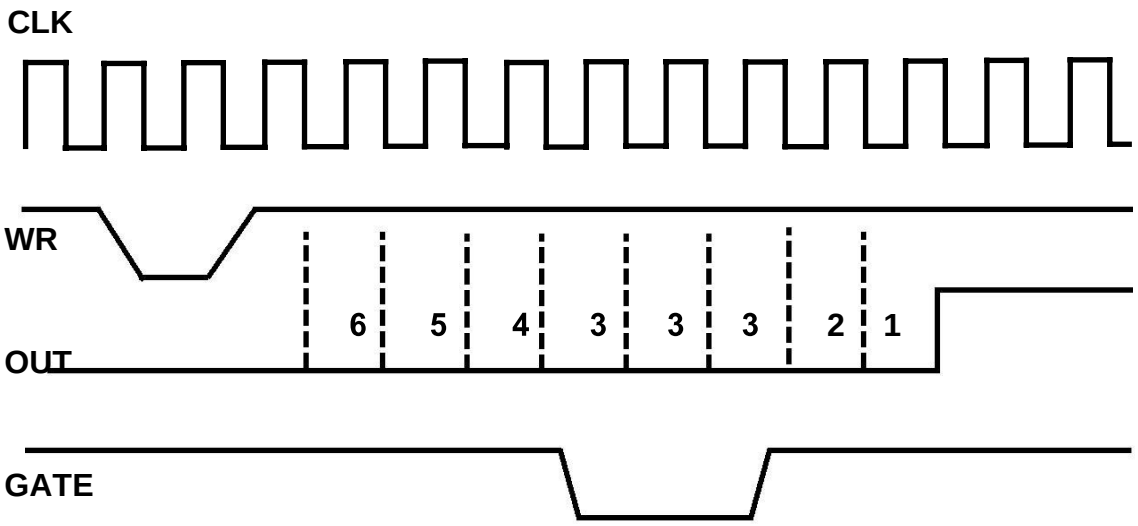


Fig 3.16 Mode 0 count when Gate is low temporarily (disabled)

Mode 1 Programmable mono-shot

The output goes low with the Gate pulse for a predetermined period depending on the counter. The counter is disabled if the GATE pulse goes momentarily low.The counter register is loaded with a count value as in the previous case (say 5). The output responds to the GATE input and goes low for period that equals the count down period of the register (5 clock pulses in this period). By changing the value of this count the duration of the output pulse can be changed. If the GATE becomes low before the count down is completed then the counter will be suspended at that state as long as GATE is low. Thus it works as a mono-shot.

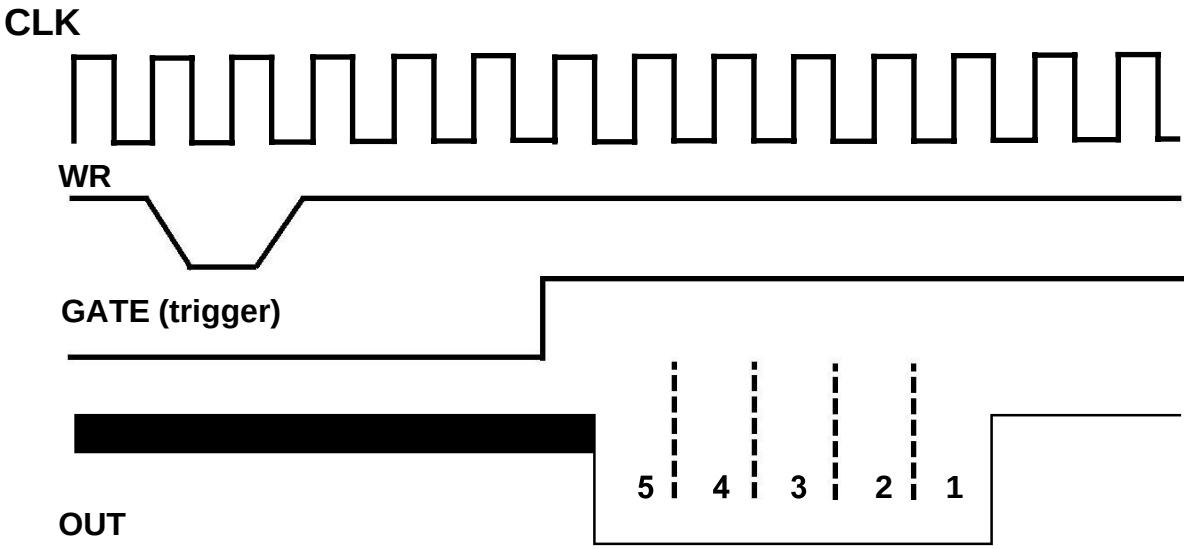


Fig 3.17 Mode 1 The Gate goes high. The output goes low for the perioddepending on the count

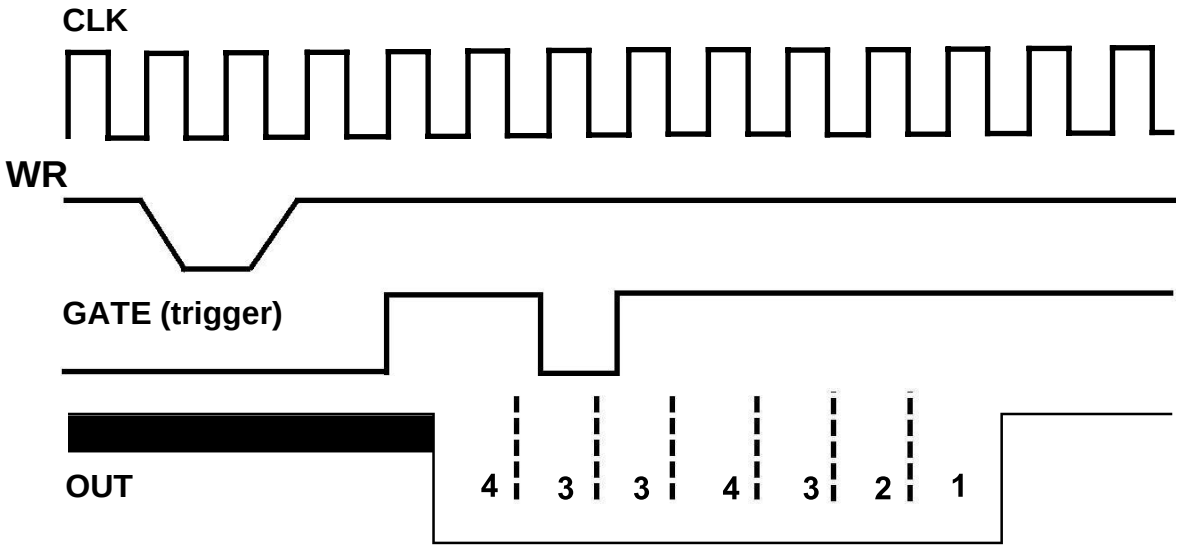


Fig 3.18 Mode 1 The Gate pulse is disabled momentarily causing the counter tostop.

Mode 2 Programmable Rate Generator

In this mode it operates as a rate generator. The output goes high for a period that equals the time of count down of the count register (3 in this case). The output goes low exactly for one clock period before it becomes high again. This is a periodic operation.

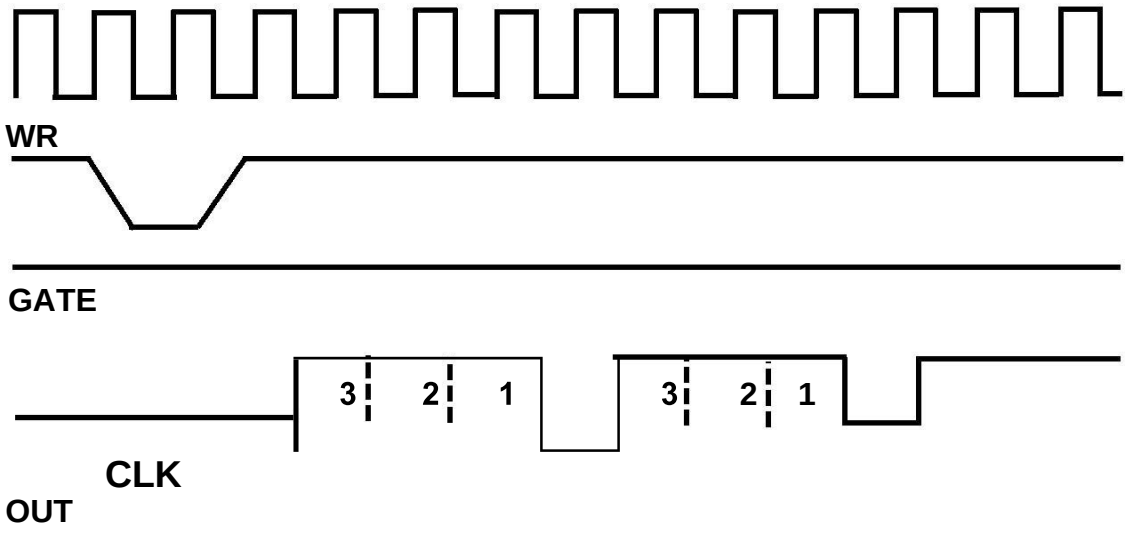


Fig 3.19 Mode 2 Operation when the GATE is kept high

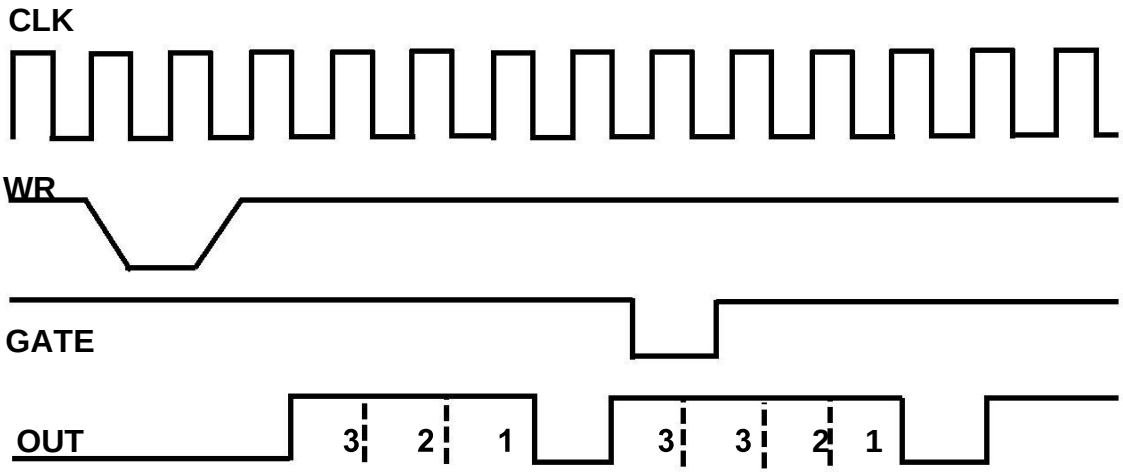


Fig 3.20 Mode 2 operation when the GATE is disabled momentarily.

Mode 3 Programmable Square Wave Rate Generator

It is similar to Mode 2 but the output high and low period is symmetrical. The output

goes high after the count is loaded and it remains high for period which equals the count down period of the counter register. The output subsequently goes low for an equal period and hence generates a symmetrical square wave unlike Mode 2. The GATE has no role here.

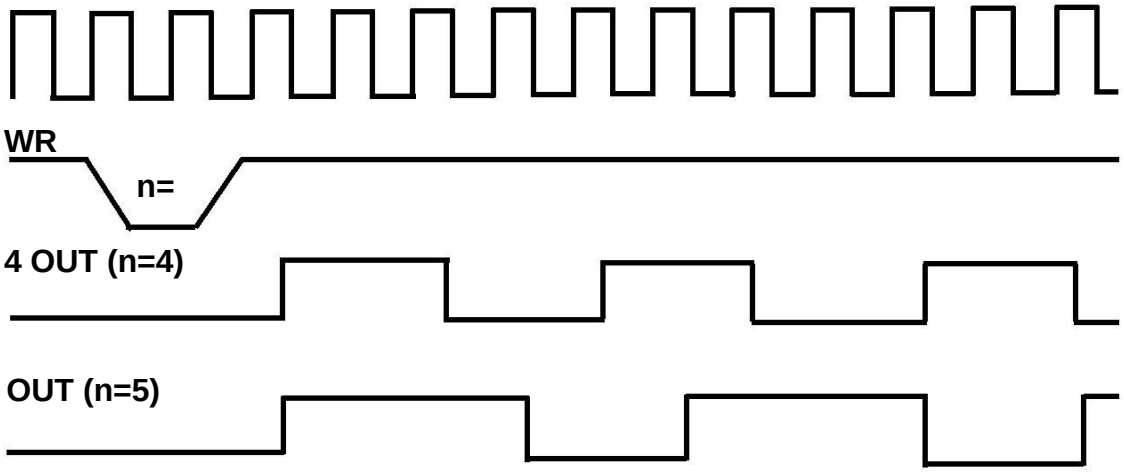


Fig 3.21 Mode3 Operation: Square Wave generator

Mode 4 Software Triggered Strobe

In this mode after the count is loaded by the processor the count down starts. The output goes low for one clock period after the count down is complete. The count down can be suspended by making the GATE low . This is also called a software triggered strobe as the count down is initiated by a program.

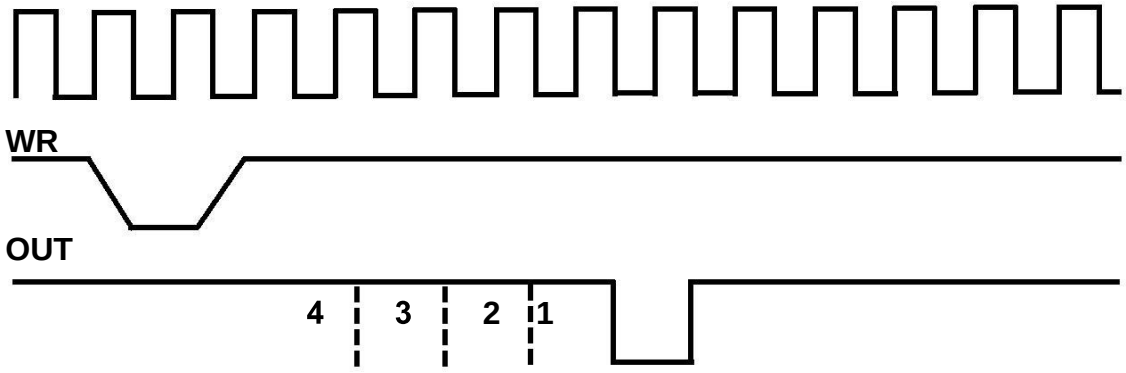


Fig 3.22 Mode 4 Software Triggered Strobe when GATE is high

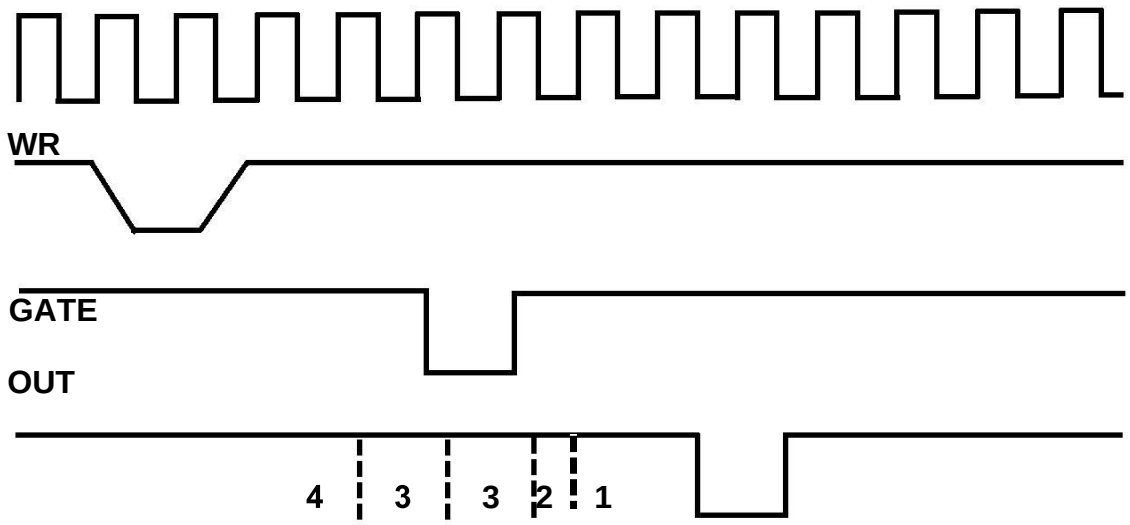


Fig 3.23 Mode 4 Software Triggered Strobe when GATE is momentarily low

Mode 5 Hardware Triggered Strobe

The count is loaded by the processor but the count down is initiated by the GATE pulse. The transition from low to high of the GATE pulse enables count down. The output goes low for one clock period after the count down is complete.

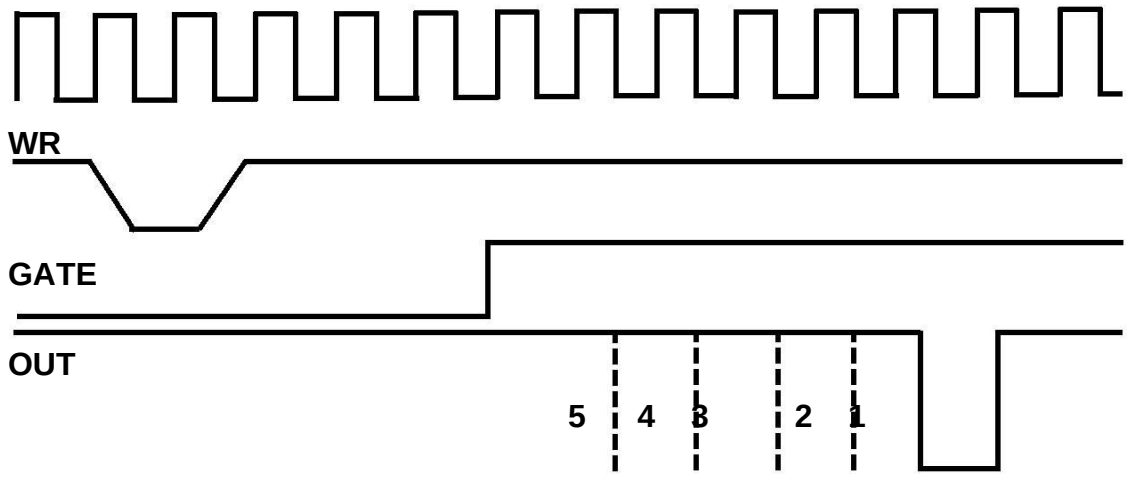


Fig 3.24 Mode 5 Hardware Triggered Strobe

PROGRAMMABLE INTERRUPT CONTROLLER-8259

FEAUTURES OF 8259

- 8086, 8088 Compatible
- MCS-80, MCS-85 Compatible
- Eight-Level Priority Controller
- Expandable to 64 Levels
- Programmable Interrupt Modes
- Individual Request Mask Capability
- Single +5V Supply (No Clocks)
- Available in 28-Pin DIP and 28-Lead PLCC Package
- Available in EXPRESS
    - Standard Temperature Range
    - Extended Temperature Range

The Intel 8259A Programmable Interrupt Controller handles up to eight vectored priority interrupts for the CPU. It is cascadable for up to 64 vectored priority interrupts without additional circuitry. It is packaged in a 28-pin DIP, uses NMOS technology and requires a single a5V supply. Circuitry is static, requiring no clock input. The 8259A is designed to minimize the software and real time overhead in handling multi-level priority interrupts. It has several modes, permitting optimization for a variety of system requirements. The 8259A is fully upward compatible with the Intel 8259. Software originally written for the 8259 will operate the 8259A in all 8259 equivalent modes (MCS-80/85, Non-Buffered, Edge Triggered). Pin Diagram of 8259 is shown in figure 17.

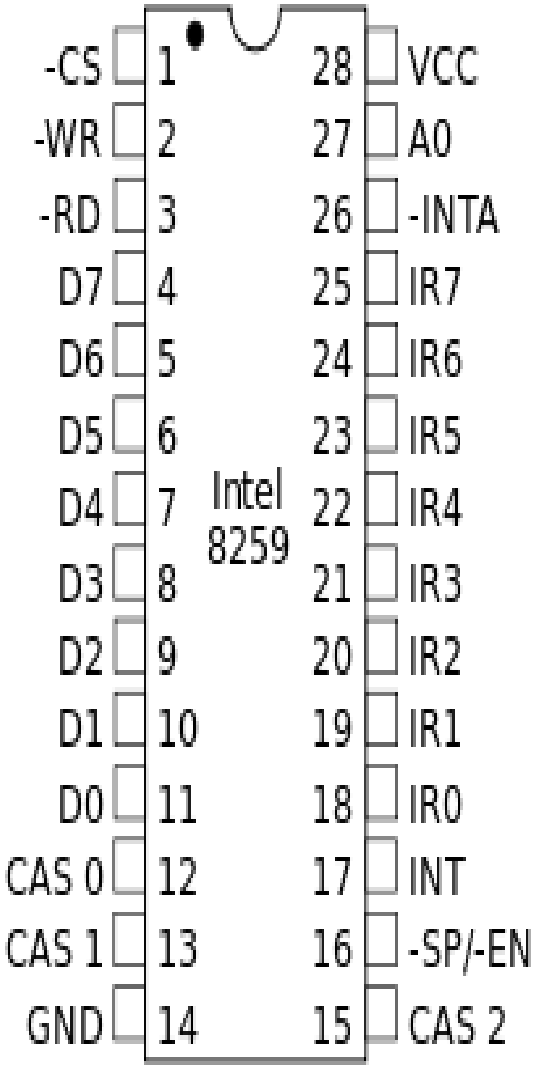


Fig.3.25 Pin Diagram of 8259

Pin Description of 8259

Symbol	Pin No.	Type	Name and Function
V_{CC}	28	I	**SUPPLY:** +5V Supply.
GND	14	I	**GROUND**
$\overline{CS}$	1	I	**CHIP SELECT:** A low on this pin enables $\overline{RD}$ and $\overline{WR}$ communication between the CPU and the 8259A. INTA functions are independent of CS.
$\overline{WR}$	2	I	**WRITE:** A low on this pin when CS is low enables the 8259A to accept command words from the CPU.
$\overline{RD}$	3	I	**READ:** A low on this pin when CS is low enables the 8259A to release status onto the data bus for the CPU.
D_7–D_0	4–11	I/O	**BIDIRECTIONAL DATA BUS:** Control, status and interrupt-vector information is transferred via this bus.
CAS_0–CAS_2	12, 13, 15	I/O	**CASCADE LINES:** The CAS lines form a private 8259A bus to control a multiple 8259A structure. These pins are outputs for a master 8259A and inputs for a slave 8259A.
$\overline{SP}/\overline{EN}$	16	I/O	**SLAVE PROGRAM/ENABLE BUFFER:** This is a dual function pin. When in the Buffered Mode it can be used as an output to control buffer transceivers (EN). When not in the buffered mode it is used as an input to designate a master (SP = 1) or slave (SP = 0).
INT	17	O	**INTERRUPT:** This pin goes high whenever a valid interrupt request is asserted. It is used to interrupt the CPU, thus it is connected to the CPU's interrupt pin.
IR_0–IR_7	18–25	I	**INTERRUPT REQUESTS:** Asynchronous inputs. An interrupt request is executed by raising an IR input (low to high), and holding it high until it is acknowledged (Edge Triggered Mode), or just by a high level on an IR input (Level Triggered Mode).
$\overline{INTA}$	26	I	**INTERRUPT ACKNOWLEDGE:** This pin is used to enable 8259A interrupt-vector data onto the data bus by a sequence of interrupt acknowledge pulses issued by the CPU.
A_0	27	I	**AO ADDRESS LINE:** This pin acts in conjunction with the $\overline{CS}$, $\overline{WR}$, and $\overline{RD}$ pins. It is used by the 8259A to decipher various Command Words the CPU writes and status the CPU wishes to read. It is typically connected to the CPU A0 address line (A1 for 8086, 8088).

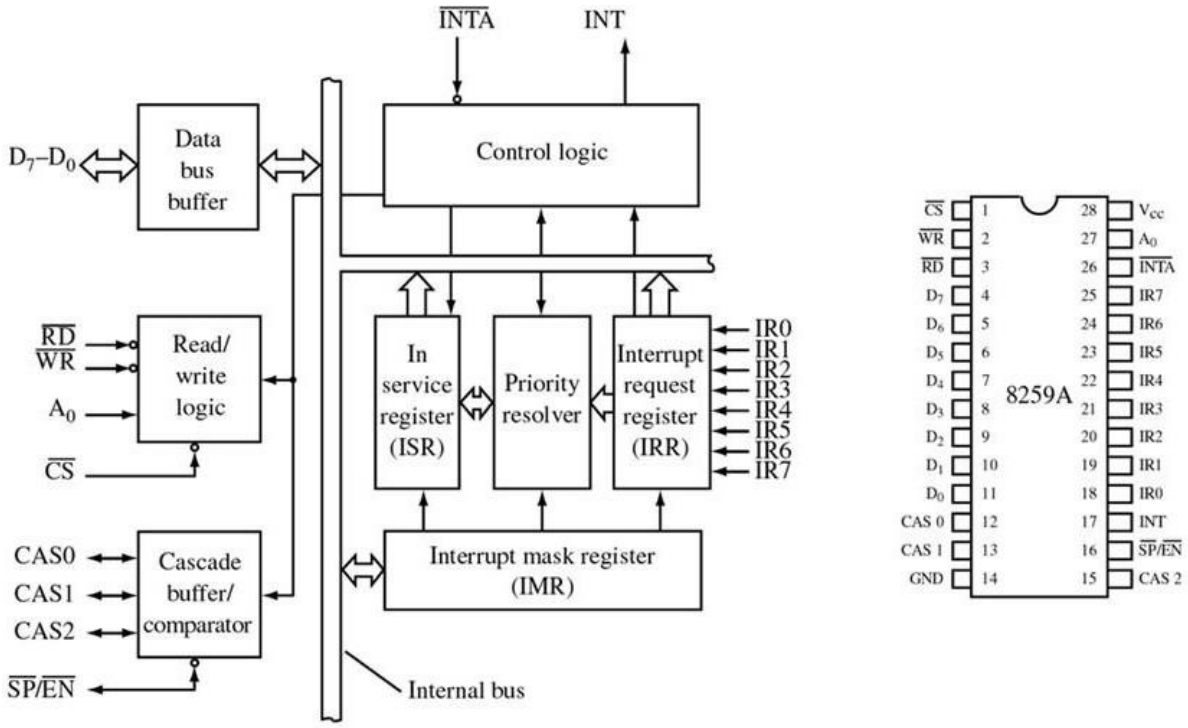


Fig 3.26 Block Diagram of 8259

A more desirable method would be one that would allow the microprocessor to be executing its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is complete, however, the processor would resume exactly where it left off. This method is called Interrupt. It is easy to see that system throughput would drastically increase, and thus more tasks could be assumed by the micro-computer to further enhance its cost effectiveness. Block Diagram of 8259 is shown in figure 18.

The Programmable Interrupt Controller (PIC) functions as an overall manager in an Interrupt-Driven system environment. It accepts requests from the peripheral equipment, determines which of the in-coming requests is of the highest importance (priori-ty), ascertains whether the incoming request has a higher priority value than the

level currently being serviced, and issues an interrupt to the CPU based on this determination.

The 8259A is a device specifically designed for use in real time, interrupt driven microcomputer systems. It manages eight levels or requests and has built-in features for expandability to other 8259A's (up to 64 levels). It is programmed by the system's software as an I/O peripheral. A selection of priority modes is available to the programmer so that the manner in which the requests are processed by the 8259A can be configured to match his system requirements. The priority modes can be changed or reconfigured dynamically at any time during the main program. This means that the complete interrupt structure can be defined as required, based on the total system environment.

INTERRUPT REQUEST REGISTER (IRR) AND IN-SERVICE REGISTER (ISR)

The interrupts at the IR input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service (ISR). The IRR is used to store all the interrupt levels which are requesting service; and the ISR is used to store all the interrupt levels which are being serviced.

PRIORITY RESOLVER

This logic block determines the priorites of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during INTA pulse.

INTERRUPT MASK REGISTER (IMR)

The IMR stores the bits which mask the interrupt lines to be masked. The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower quality.

INT (INTERRUPT)

This output goes directly to the CPU interrupt input. The VOH level on this line is designed to be fully compatible with the 8080A, 8085A and 8086 input levels.

INTA (INTERRUPT ACKNOWLEDGE)

INTA pulses will cause the 8259A to release vectoring information onto the data bus. The format of this data depends on the system mode (mPM) of the 8259A.

DATA BUS BUFFER

This 3-state, bidirectional 8-bit buffer is used to inter-face the 8259A to the system Data Bus. Control words and status information are transferred through the Data Bus Buffer.

READ/WRITE CONTROL LOGIC

The function of this block is to accept Output commands from the CPU. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW) registers which store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the Data Bus.

CS (CHIP SELECT)

A LOW on this input enables the 8259A. No reading or writing of the chip will occur unless the device is selected.

WR (WRITE)

A LOW on this input enables the CPU to write con-trol words (ICWs and OCWs) to the 8259A.

RD (READ)

A LOW on this input enables the 8259A to send the status of the Interrupt Request Register (IRR), In Service Register (ISR), the Interrupt Mask Register (IMR), or the Interrupt level onto the Data Bus.

A0

This input signal is used in conjunction with WR and RD signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines.

INTERRUPT SEQUENCE

The powerful features of the 8259A in a microcomputer system are its programmability and the interrupt routine addressing capability. The latter allows direct or indirect jumping to the specific interrupt routine requested without any polling of the interrupting devices. The normal sequence of events during an interrupt depends on the type of CPU being used.

The events occur as follows in an MCS-80/85 sys-tem:

1. One or more of the INTERRUPT REQUEST lines (IR7±0) are raised high, setting the correspond-ing IRR bit(s).
2. The 8259A evaluates these requests, and sends an INT to the CPU, if appropriate.
3. The CPU acknowledges the INT and responds with an INTA pulse.
4. Upon receiving an INTA from the CPU group, the highest priority ISR bit is set, and the correspond-ing IRR bit is reset. The 8259A will also release a CALL instruction code (11001101) onto the 8-bit Data Bus through its D7±0 pins.
5. This CALL instruction will initiate two more INTA pulses to be sent to the 8259A from the CPU group.
6. These two INTA pulses allow the 8259A to re-lease its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first INTA pulse and the higher 8-bit address is released at the second INTA pulse.
7. This completes the 3-byte CALL instruction re-leased by the 8259A. In the AEOI mode the ISR bit is reset at the end of the third INTA pulse. Otherwise, the ISR

bit remains set until an appropriate EOI command is issued at the end of the interrupt sequence.

8. The events occurring in an 8086 system are the same until step 4.
9. Upon receiving an INTA from the CPU group, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive the Data Bus during this cycle.

10. The 8086 will initiate a second INTA pulse. During this pulse, the 8259A releases an 8-bit pointer onto the Data Bus where it is read by the CPU.

11. This completes the interrupt cycle. In the AEOI mode the ISR bit is reset at the end of the second INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt subroutine.

If no interrupt request is present at step 4 of either sequence (i.e., the request was too short in duration) the 8259A will issue an interrupt level 7. Both the vectoring bytes and the CAS lines will look like an interrupt level 7 was requested.

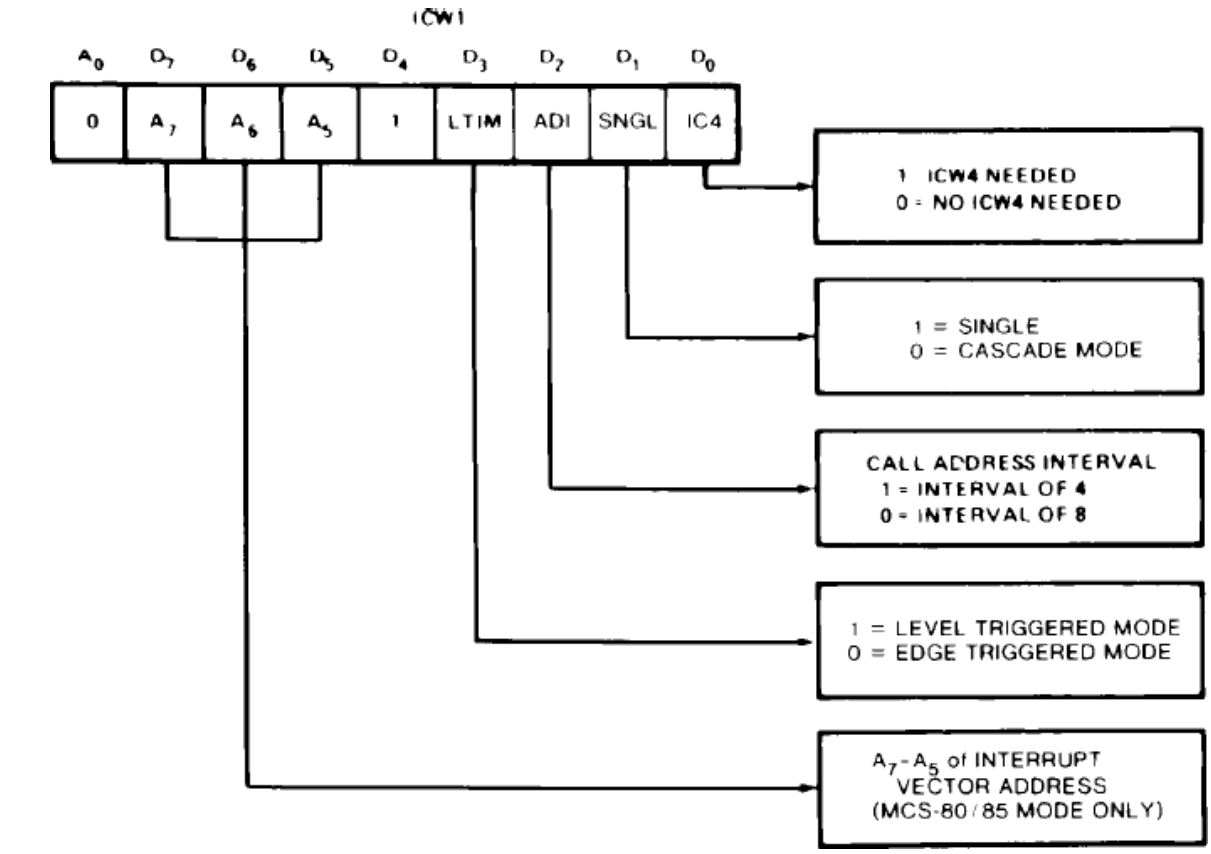
When the 8259A PIC receives an interrupt, INT be-comes active and an interrupt acknowledge cycle is started. If a higher priority interrupt occurs between the two INTA pulses, the INT line goes inactive immediately after the second INTA pulse. After an un-specified amount of time the INT line is activated again to signify the higher priority interrupt waiting for service. This inactive time is not specified and can vary between parts. The designer should be aware of this consideration when designing a sys-tem which uses the 8259A. It is recommended that proper asynchronous design techniques be followed.

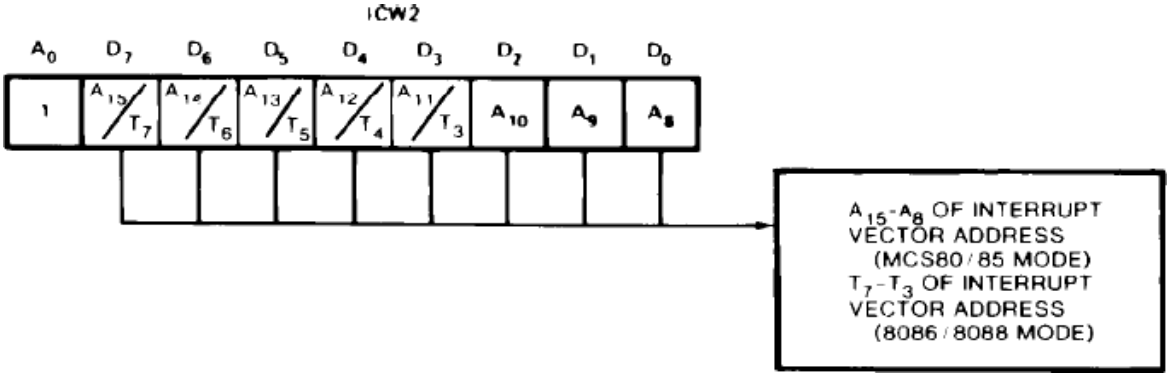
INITIALIZATION COMMAND WORDS

Whenever a command is issued with A0 e 0 and D4 e 1, this is interpreted as Initialization Command Word 1 (ICW1). ICW1 starts the initialization sequence during which the following automatically occur.

a. The edge sense circuit is reset, which means that following initialization, an interrupt request (IR) input must make a low-to-high transition to generate an interrupt.
b. The Interrupt Mask Register is cleared.
c. IR7 input is assigned priority 7.
d. The slave mode address is set to 7.
e. Special Mask Mode is cleared and Status Read isset to IRR.
f. If IC4 e 0, then all functions selected in ICW4are set to zero. (Non-Buffered mode*, no Auto-EOI, MCS-80, 85 system).

Initialization Command Word Format is as shown in figure 19.


ICW1
A0 D7 D6 D5 D4 D3 D2 D1 D0
0 A7 A6 A5 1 LTIM ADI SNGL IC4
1 ICW4 NEEDED
0 = NO ICW4 NEEDED
1 = SINGLE
0 = CASCADE MODE
CALL ADDRESS INTERVAL
1 = INTERVAL OF 4
0 = INTERVAL OF 8
1 = LEVEL TRIGGERED MODE
0 = EDGE TRIGGERED MODE
A7-A5 of INTERRUPT
VECTOR ADDRESS
(MCS-80/85 MODE ONLY)


ICW2
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 A15/T7 A14/T6 A13/T5 A12/T4 A11/T3 A10 A9 A8
A15-A8 OF INTERRUPT
VECTOR ADDRESS
(MCS80/85 MODE)
T7-T3 OF INTERRUPT
VECTOR ADDRESS
(8086/8088 MODE)

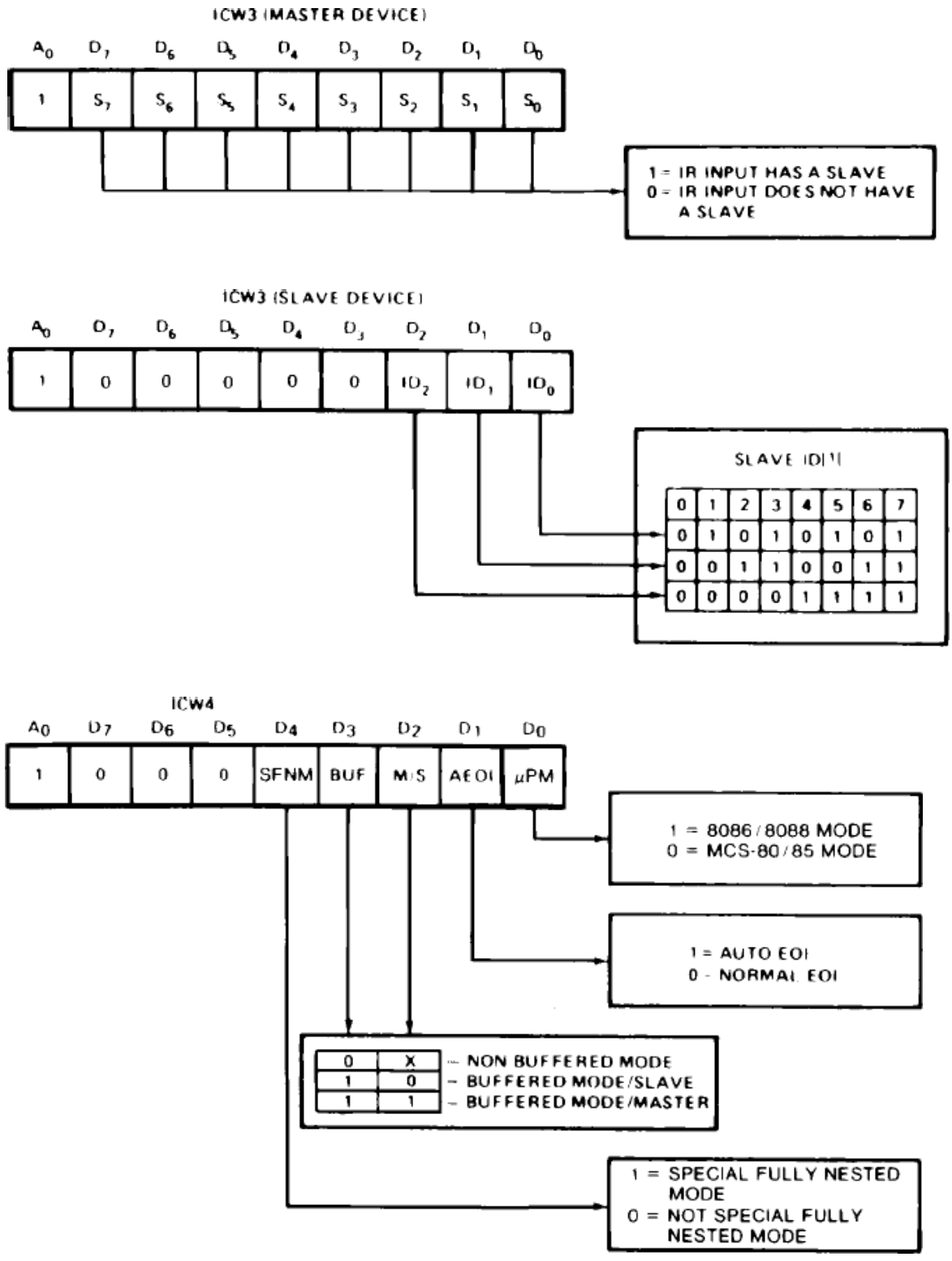


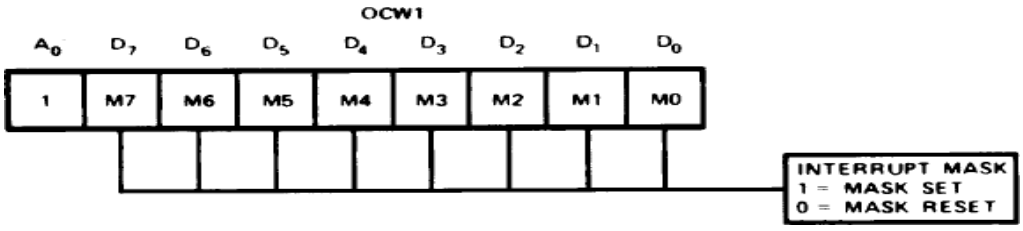
Fig 3.27. Initialization Command Word Format

OPERATION COMMAND WORDS

After the Initialization Command Words (ICWs) are programmed into the 8259A, the chip is ready to accept interrupt requests at its input lines. However, during the 8259A operation, a selection of algorithms can command the 8259A to operate in various modes through the Operation Command Words (OCWs). Operation Command Word format is as shown in figure 20

		OCW1							
A0		**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
1		M7	M6	M5	M4	M3	M2	M1	M0
		OCW2							
0		R	SL	EOI	0	0	L2	L1	L0
		OCW3							
0		0	ESMM	SMM	0	1	P	RR	RIS

Fig 3.28 Operational Control Words



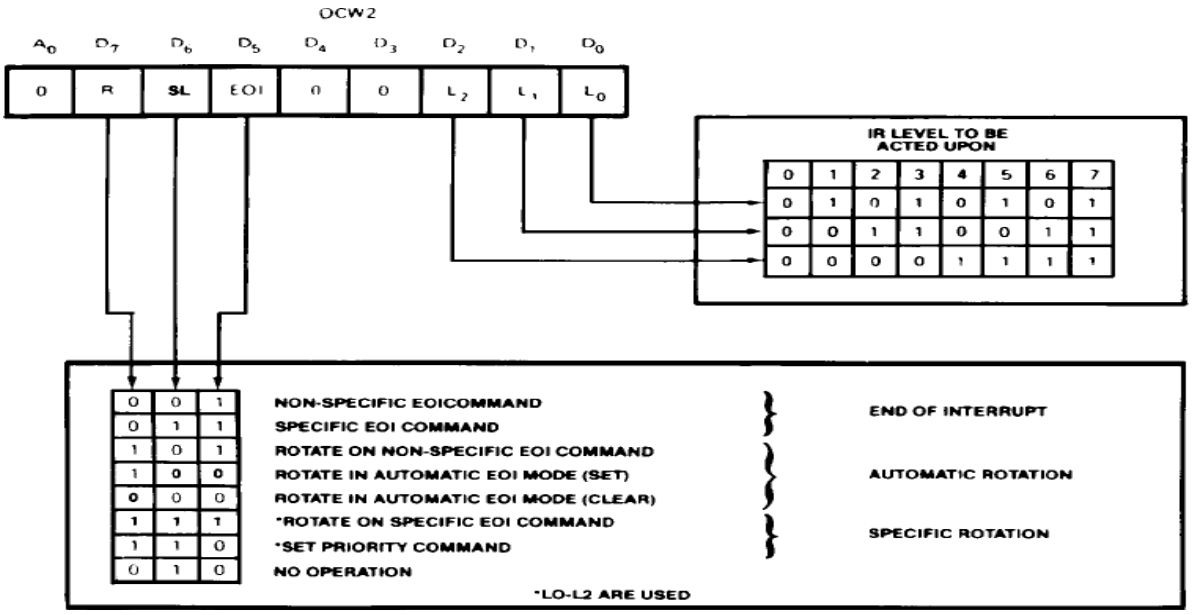


Fig 3.29 Operation Command Word Format

DMA CONTROLLER 8257

The Direct Memory Access or DMA mode of data transfer is the fastest amongstall the modes of data transfer. In this mode, the device may transfer data directly to/from memory without any interference from the CPU. The device requests the CPU (through aDMA controller) to hold its data, address and control bus, so that the device may transfer data directly to/from memory. The DMA data transfer is initiated only after receiving HLDA signal from the CPU. Intel's 8257 is a four channel DMA controller designed to be interfaced with their family of microprocessors. The 8257, on behalf of the devices, requests the CPU for bus access using local bus request input i.e. HOLD in minimum mode. In maximum mode of the microprocessor RQ/GT pin is used as bus request input. On receiving the HLDA signal (in minimum mode) or RQ/GT signal (in maximum mode) from the CPU, the requesting devices gets the access of the bus, and it completes the required number of DMA cycles for the data transfer and then hands over the control of the bus back to the CPU.

Internal Architecture of 8257

The internal architecture of 8257 is shown in figure. The chip support four DMA channels, i.e. four peripheral devices can independently request for DMA data transfer

through these channels at a time. The DMA controller has 8-bit internal data buffer, a read/write unit, a control unit, a priority resolving unit along with a set of registers.

The 8257 performs the DMA operation over four independent DMA channels. Each of four channels of 8257 has a pair of two 16-bit registers, viz. DMA address register and terminal count register.

There are two common registers for all the channels, namely, mode set register and status register. Thus there are a total of ten registers. The CPU selects one of these ten registers using address lines Ao-A3. Table shows how the Ao-A3 bits may be used for selecting one of these registers.

DMA ADDRESS REGISTER

Each DMA channel has one DMA address register. The function of this register is to store the address of the starting memory location, which will be accessed by the DMA channel. Thus the starting address of the memory block which will be accessed by the device is first loaded in the DMA address register of the channel. The device that wants to transfer data over a DMA channel, will access the block of the memory with the starting address stored in the DMA Address Register.

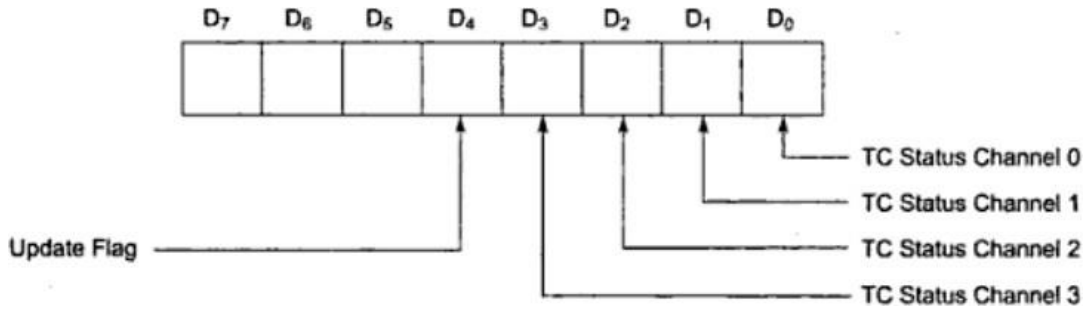
TERMINAL COUNT REGISTER

Each of the four DMA channels of 8257 has one terminal count register (TC). This 16-bit register is used for ascertaining that the data transfer through a DMA channel ceases or stops after the required number of DMA cycles. The low order 14-bits of the terminal count register are initialized with the binary equivalent of the number of required DMA cycles minus one. After each DMA cycle, the terminal count register content will be decremented by one and finally it becomes zero after the required number of DMA cycles are over. The bits14 and 15 of this register indicate the type of the DMA operation (transfer). If the device wants to write data into the memory, the DMA operation is called DMA write operation. Bit 14 of the register in this case will be set to one and bit 15 will be set to zero.

STATUS REGISTER

The status register of 8257 is shown in figure. The lower order 4-bits of this register contain the terminal count status for the four individual channels. If any of these bits is set, it indicates that the specific channel has reached the terminal count condition.

These bits remain set till either the status is read by the CPU or the 8257 is reset. The update flag is not affected by the read operation. This flag can only be cleared by resetting 8257 or by resetting the auto load bit of the mode set register. If the update flag is set, the contents of the channel 3 registers are reloaded to the corresponding registers of channel 2 whenever the channel 2 reaches a terminal count condition, after transferring one block and the next block is to be transferred using the autoload feature of 8257.



The update flag is set every time, the channel 2 registers are loaded with contents of the channel 3 registers. It is cleared by the completion of the first DMA cycle of the new block. This register can only read.

DATA BUS BUFFER, READ/WRITE LOGIC, CONTROL UNIT AND PRIORITY RESOLVER

The 8-bit. Tristate, bidirectional buffer interfaces the internal bus of 8257 with the external system bus under the control of various control signals. In the slave mode, the read/write logic accepts the I/O Read or I/O Write signals, decodes the Ao-A3 lines and either writes the contents of the data bus to the addressed internal register or reads the

contents of the selected register depending upon whether IOW or IOR signal is activated.

In master mode, the read/write logic generates the IOR and IOW signals to control the data flow to or from the selected peripheral. The control logic controls the sequences of operations and generates the required control signals like AEN, ADSTB, MEMR, MEMW, TC and MARK along with the address lines A4-A7, in master mode. The priority resolver resolves the priority of the four DMA channels depending upon whether normal priority or rotating priority is programmed.

Signal Description of 8257

DRQ0-DRQ3

These are the four individual channel DMA request inputs, used by the peripheral devices for requesting the DMA services. The DRQ0 has the highest priority while DRQ3 has the lowest one, if the fixed priority mode is selected.

DACK0-DACK3:

These are the active-low DMA acknowledge output lines which inform the requesting peripheral that the request has been honoured and the bus is relinquished by the CPU. These lines may act as strobe lines for the requesting devices.

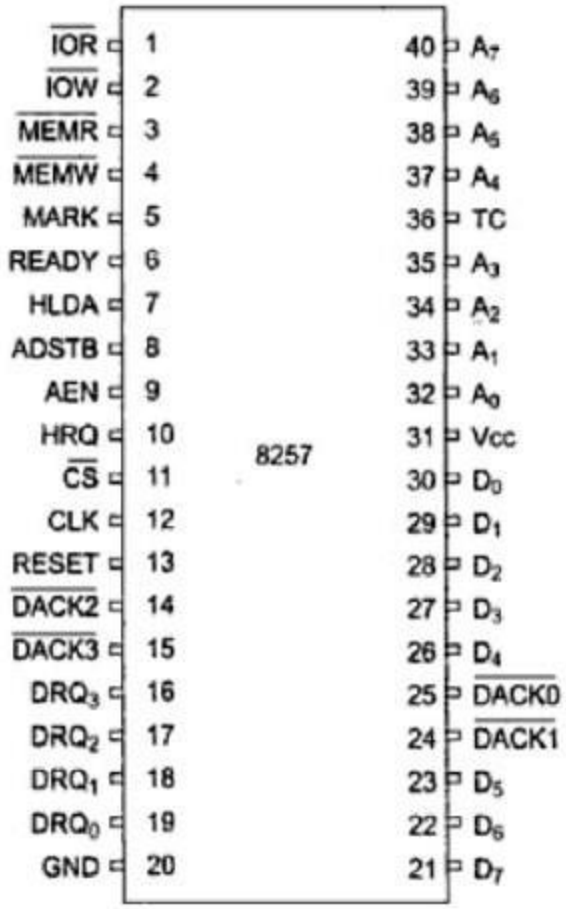


Fig 3.30 Pin Description of 8257

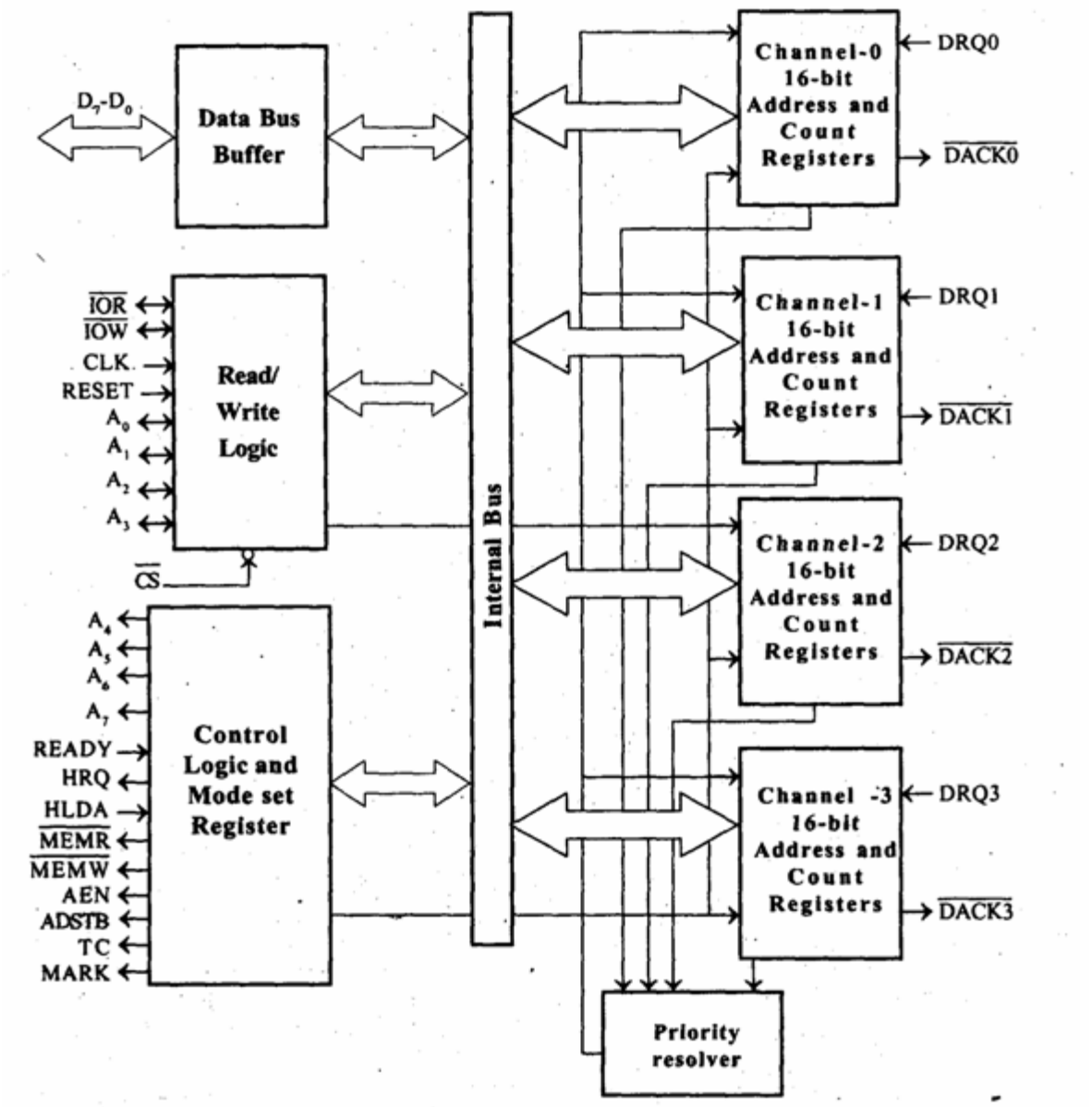


Fig 3.31 Architecture of 8257

Do-D7:

These are bidirectional, data lines used to interface the system bus with theinternal data bus of 8257. These lines carry command words to 8257 and status wordfrom 8257, in slave mode, i.e. under the control of CPU.The data over these lines may be transferred in both the directions. When the 8257 is thebus master (master mode, i.e. not under

CPU control), it uses Do-D7 lines to send higherbyte of the generated address to the latch. This address is further latched using ADSTBsignal. the address is transferred over Do-D7 during the first clock cycle of the DMAcycle. During the rest of the period, data is available on the data bus.

IOR:

This is an active-low bidirectional tristate input line that acts as an input in theslave mode. In slave mode, this input signal is used by the CPU to read internal registersof 8257.this line acts output in master mode. In master mode, this signal is used to readdata from a peripheral during a memory write cycle.

IOW:

This is an active low bidirection tristate line that acts as input in slave mode to load the contents of the data bus to the 8-bit mode register or upper/lower byte of a 16-bit DMA address register or terminal count register. In the master mode, it is a control output that loads the data to a peripheral during DMA memory read cycle (write to peripheral).

CLK:

This is a clock frequency input required to derive basic system timings for theinternal operation of 8257.

RESET :

This active-high asynchronous input disables all the DMA channels by clearing the mode register and tristates all the control lines.

Ao-A3:

These are the four least significant address lines. In slave mode, they act as input which select one of the registers to be read or written. In the master mode, they are the four least significant memory address output lines generated by 8257.

CS:

This is an active-low chip select line that enables the read/write operations from/to 8257, in slave mode. In the master mode, it is automatically disabled to prevent the chip from getting selected (by CPU) while performing the DMA operation.

A4-A7:

This is the higher nibble of the lower byte address generated by 8257 during the master mode of DMA operation.

READY:

This is an active-high asynchronous input used to stretch memory read and writecycles of 8257 by inserting wait states. This is used while interfacing slower peripherals.

HRQ:

The hold request output requests the access of the system bus. In the noncascaded8257 systems, this is connected with HOLD pin of CPU. In the cascade mode, this pin of a slave is connected with a DRQ input line of the master 8257, while that of the master is connected with HOLD input of the CPU.

HLDA :

The CPU drives this input to the DMA controller high, while granting the bus tothe device. This pin is connected to the HLDA output of the CPU. This input, if high, indicates to the DMA controller that the bus has been granted to the requesting peripheral by the CPU.

MEMR:

This active –low memory read output is used to read data from the addressed memory locations during DMA read cycles.

MEMW :

This active-low three state output is used to write data to the addressed memory location during DMA write operation.

ADST :

This output from 8257 strobes the higher byte of the memory address generated by the DMA controller into the latches.

AEN:

This output is used to disable the system data bus and the control the bus driven by the CPU, this may be used to disable the system address and data bus by using the enable input of the bus drivers to inhibit the non-DMA devices from responding during DMA operations. If the 8257 is I/O mapped, this should be used to disable the other I/O devices, when the DMA controller addresses is on the address bus.

TC:

Terminal count output indicates to the currently selected peripherals that thepresent DMA cycle is the last for the previously programmed data block. If the TC STOP bit in the mode set register is set, the selected channel will be disabled at the end of the DMA cycle. The TC pin is activated when the 14-bit content of the terminal count register of the selected channel becomes equal to zero. The lower order 14 bits of the terminal count register are to be programmed with a 14-bit equivalent of (n-1), if n is the desired number of DMA cycles.

MARK:

The modulo 128 mark output indicates to the selected peripheral that the current DMA cycle is the 128th cycle since the previous MARK output. The mark will be activated after each 128 cycles or integral multiples of it from the beginning if the data block (the first DMA cycle), if the total number of the required DMA cycles (n) is completely divisible by 128.

Vcc:

This is a +5v supply pin required for operation of the circuit.

GND:

This is a return line for the supply (ground pin of the IC).

DMA CONTROLLER 8257

The Direct Memory Access or DMA mode of data transfer is the fastest amongst all the modes of data transfer. In this mode, the device may transfer data directly to/from memory without any interference from the CPU. The device requests the CPU (through a DMA controller) to hold its data, address and control bus, so that the device may transfer data directly to/from memory. The DMA data transfer is initiated only after receiving HLDA signal from the CPU. Intel's 8257 is a four channel DMA controller designed to be interfaced with their family of microprocessors. The 8257, on behalf of the devices, requests the CPU for bus access using local bus request input i.e. HOLD in minimum mode. In maximum mode of the microprocessor RQ/GT pin is used as bus request input. On receiving the HLDA signal (in minimum mode) or RQ/GT signal (in maximummode) from the CPU, the requesting devices gets the access of the bus, and it completes the required number of DMA cycles for the data transfer and then hands over the control of the bus back to the CPU.

Internal Architecture of 8257

The internal architecture of 8257 is shown in figure. The chip support four DMA channels, i.e. four peripheral devices can independently request for DMA data transfer through these channels at a time. The DMA controller has 8-bit internal data buffer, a read/write unit, a control unit, a priority resolving unit along with a set of registers.

The 8257 performs the DMA operation over four independent DMA channels. Each of four channels of 8257 has a pair of two 16-bit registers, viz. DMA address register and terminal count register.

There are two common registers for all the channels, namely, mode set register and status register. Thus there are a total of ten registers. The CPU selects one of these ten registers using address lines Ao-A3. Table shows how the Ao-A3 bits may be used for selecting one of these registers.

DMA ADDRESS REGISTER

Each DMA channel has one DMA address register. The function of this register is to store the address of the starting memory location, which will be accessed by the DMA channel. Thus the starting address of the memory block which will be accessed by the device is first loaded in the DMA address register of the channel. The device that wants to transfer data over a DMA channel, will access the block of the memory with the starting address stored in the DMA Address Register.

TERMINAL COUNT REGISTER

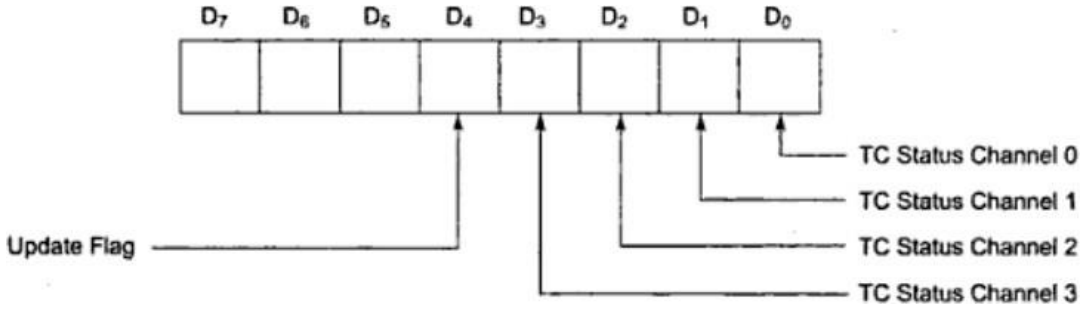
Each of the four DMA channels of 8257 has one terminal count register (TC). This 16-bit register is used for ascertaining that the data transfer through a DMA channel ceases or stops after the required number of DMA cycles. The low order 14-bits of the terminal count register are initialized with the binary equivalent of the number of required DMA cycles minus one. After each DMA cycle, the terminal count register content will be decremented by one and finally it becomes zero after the required number of DMA cycles are over. The bits14 and 15 of this register indicate the type of the DMA operation (transfer). If the device wants to write data into the memory, the DMA operation is called DMA write operation. Bit 14 of the register in this case will be set to one and bit 15 will be set to zero.

STATUS REGISTER

The status register of 8257 is shown. The lower order 4-bits of this register contain the terminal count status for the four individual channels. If any of these bits is set, it indicates that the specific channel has reached the terminal count condition.

These bits remain set till either the status is read by the CPU or the 8257 is reset. The update flag is not affected by the read operation. This flag can only be cleared by

resetting 8257 or by resetting the auto load bit of the mode set register. If the update flag is set, the contents of the channel 3 registers are reloaded to the corresponding registers of channel 2 whenever the channel 2 reaches a terminal count condition, after transferring one block and the next block is to be transferred using the autoload feature of 8257.



The update flag is set every time, the channel 2 registers are loaded with contents of the channel 3 registers. It is cleared by the completion of the first DMA cycle of the newblock. This register can only read.

DATA BUS BUFFER, READ/WRITE LOGIC, CONTROL UNIT AND PRIORITY RESOLVER

The 8-bit. Tristate, bidirectional buffer interfaces the internal bus of 8257 with the external system bus under the control of various control signals. In the slave mode, the read/write logic accepts the I/O Read or I/O Write signals, decodes the Ao-A3 lines and either writes the contents of the data bus to the addressed internal register or reads the contents of the selected register depending upon whether IOW or IOR signal is activated.

In master mode, the read/write logic generates the IOR and IOW signals to control the data flow to or from the selected peripheral. The control logic controls the sequences of operations and generates the required control signals like AEN, ADSTB, MEMR, MEMW, TC and MARK along with the address lines A4-A7, in master mode. The priority resolver resolves the priority of the four DMA channels depending upon whether normal priority or rotating priority is programmed.

Signal Description of 8257

DRQ0-DRQ3

These are the four individual channel DMA request inputs, used by the peripheral devices for requesting the DMA services. The DRQ0 has the highest priority while DRQ3 has the lowest one, if the fixed priority mode is selected.

DACK0-DACK3:

These are the active-low DMA acknowledge output lines which inform the requesting peripheral that the request has been honoured and the bus is relinquished by the CPU. These lines may act as strobe lines for the requesting devices.

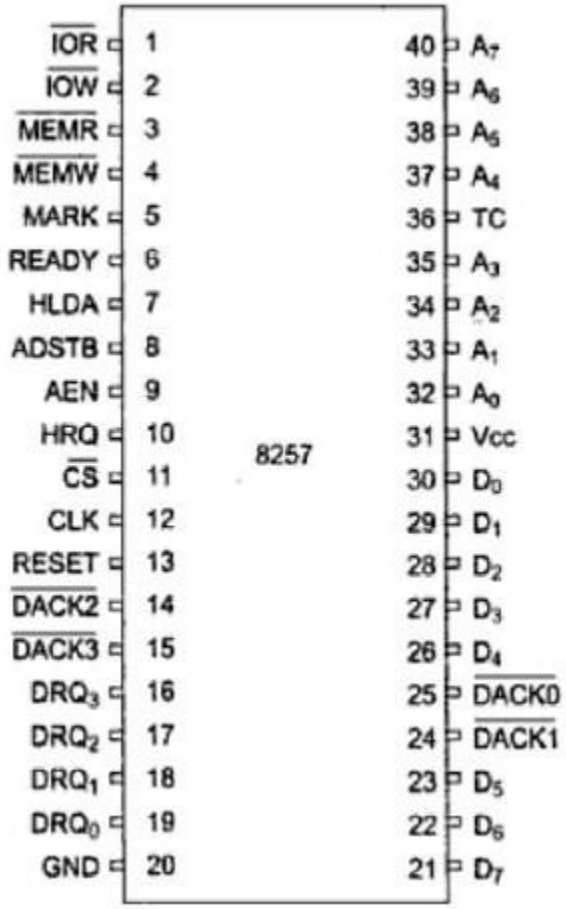


Fig.3.32 Pin Description of 8257

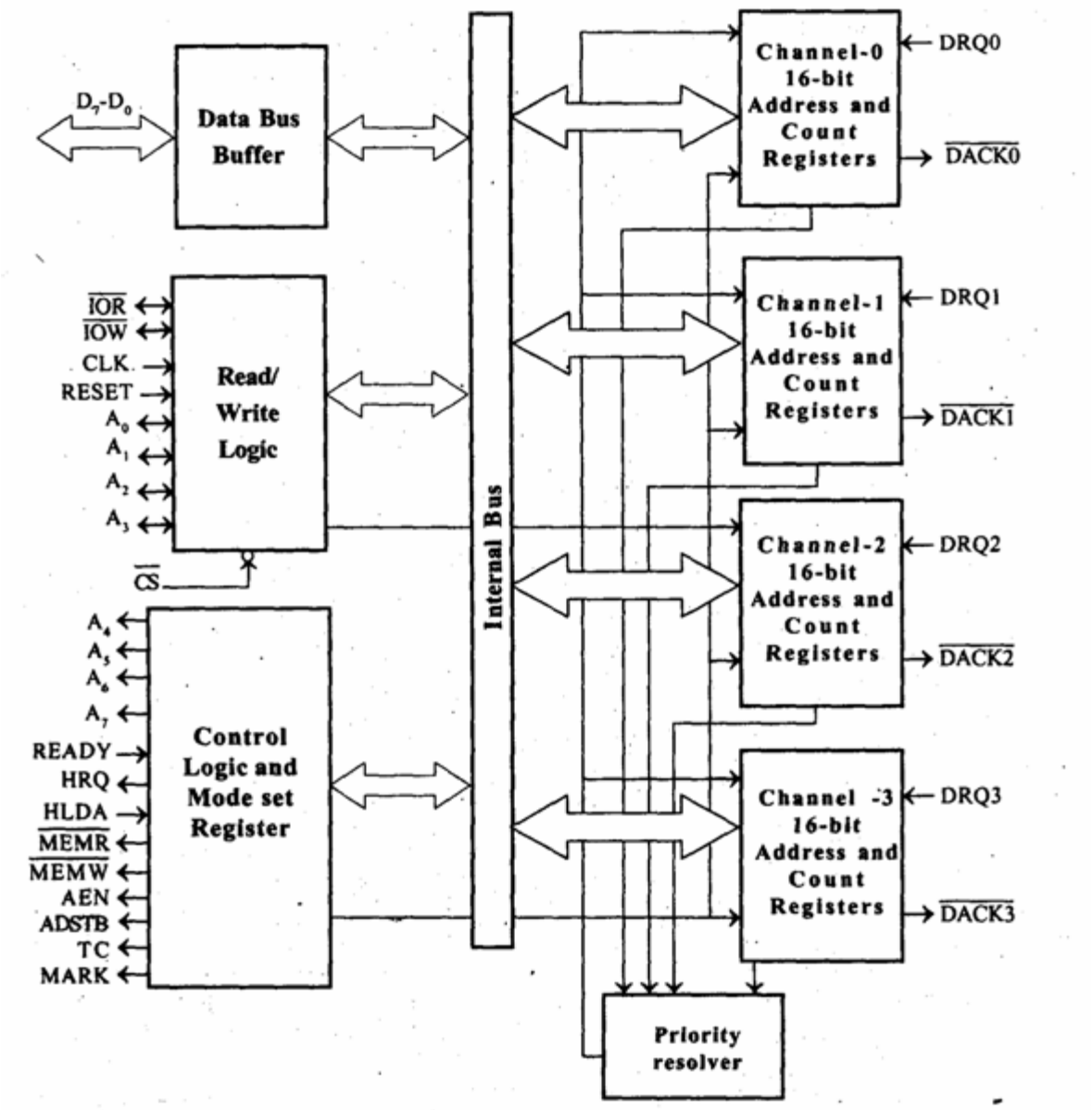


Fig 3.33 Architecture of 8257

Do-D7:

These are bidirectional, data lines used to interface the system bus with the internal data bus of 8257. These lines carry command words to 8257 and status word from 8257, in slave mode, i.e. under the control of CPU. The data over these lines may be transferred in both the directions. When the 8257 is the bus master (master mode, i.e.

not under CPU control), it uses Do-D7 lines to send higher byte of the generated address to the latch. This address is further latched using ADSTB signal. the address is transferred over Do-D7 during the first clock cycle of the DMA cycle. During the rest of the period, data is available on the data bus.

IOR:

This is an active-low bidirectional tristate input line that acts as an input in the slave mode. In slave mode, this input signal is used by the CPU to read internal registers of 8257.this line acts output in master mode. In master mode, this signal is used to read data from a peripheral during a memory write cycle.

IOW:

This is an active low bidirection tristate line that acts as input in slave mode to load the contents of the data bus to the 8-bit mode register or upper/lower byte of a 16-bit DMA address register or terminal count register. In the master mode, it is a control output that loads the data to a peripheral during DMA memory read cycle (write to peripheral).

CLK:

This is a clock frequency input required to derive basic system timings for the internal operation of 8257.

RESET :

This active-high asynchronous input disables all the DMA channels by clearing the mode register and tristates all the control lines.

Ao-A3:

These are the four least significant address lines. In slave mode, they act as input which select one of the registers to be read or written. In the master mode, they are the four least significant memory address output lines generated by 8257.

CS:

This is an active-low chip select line that enables the read/write operations from/to 8257, in slave mode. In the master mode, it is automatically disabled to preventthe chip from getting selected (by CPU) while performing the DMA operation.

A4-A7:

This is the higher nibble of the lower byte address generated by 8257 during the master mode of DMA operation.

READY:

This is an active-high asynchronous input used to stretch memory read and write cycles of 8257 by inserting wait states. This is used while interfacing slower peripherals.

HRQ:

The hold request output requests the access of the system bus. In the noncascaded8257 systems, this is connected with HOLD pin of CPU. In the cascade mode, this pin of a slave is connected with a DRQ input line of the master 8257, while that of the master is connected with HOLD input of the CPU.

HLDA :

The CPU drives this input to the DMA controller high, while granting the bus to the device. This pin is connected to the HLDA output of the CPU. This input, if high, indicates to the DMA controller that the bus has been granted to the requesting peripheral by the CPU.

MEMR:

This active –low memory read output is used to read data from the addressed memory locations during DMA read cycles.

MEMW :

This active-low three state output is used to write data to the addressed memory location during DMA write operation.

ADST :

This output from 8257 strobes the higher byte of the memory address generated by the DMA controller into the latches.

AEN:

This output is used to disable the system data bus and the control the bus driven by the CPU, this may be used to disable the system address and data bus by using the enable input of the bus drivers to inhibit the non-DMA devices from responding during DMA operations. If the 8257 is I/O mapped, this should be used to disable the other I/O devices, when the DMA controller addresses is on the address bus.

TC:

Terminal count output indicates to the currently selected peripherals that the present DMA cycle is the last for the previously programmed data block. If the TC STOP bit in the mode set register is set, the selected channel will be disabled at the end of the DMA cycle. The TC pin is activated when the 14-bit content of the terminal count register of the selected channel becomes equal to zero. The lower order 14 bits of the terminal count register are to be programmed with a 14-bit equivalent of (n-1), if n is the desired number of DMA cycles.

MARK:

The modulo 128 mark output indicates to the selected peripheral that the current DMA cycle is the 128th cycle since the previous MARK output. The mark will be activated after each 128 cycles or integral multiples of it from the beginning if the data

block (the first DMA cycle), if the total number of the required DMA cycles (n) is completely divisible by 128.

Vcc:

This is a +5v supply pin required for operation of the circuit.

GND:

This is a return line for the supply (ground pin of the IC).

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

SCHOOL OF ELECTRICAL AND ELECTRONICS
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

UNIT 4
Microprocessor , Interfacing and Its Applications-SECA1508

Keyboard/Display Controller

The Intel 8279 is a Keyboard/Display Controller is specially developed for interfacing keyboard and display devices for the Intel 8085, 8086 and 8088 microprocessors. Its important features are:

- Simultaneous keyboard and display operations.
- Scanned keyboard mode.
- Scanned sensor mode.
- 8-character keyboard FIFO.
- Right or left entry 16-byte display RAM.
- Programmable scan timing.

I. Pin details of 8279

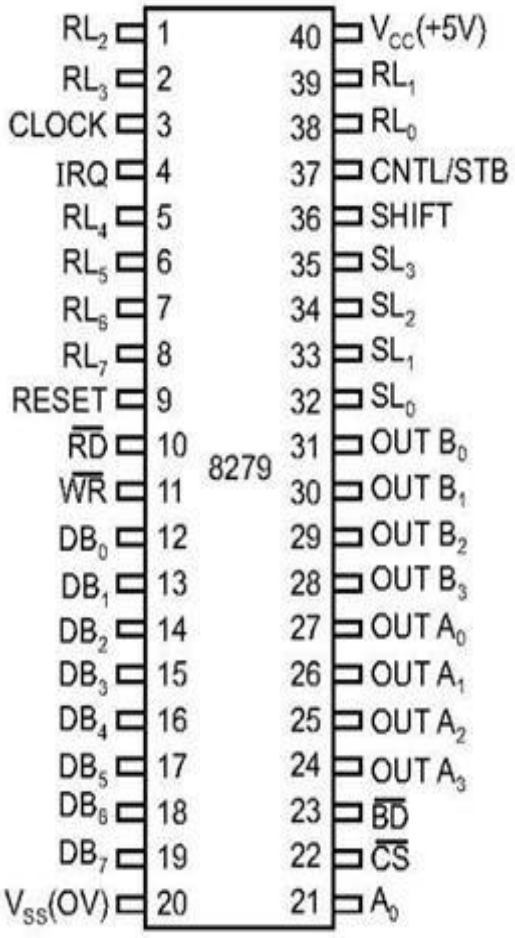


Figure 4.1 : Pin diagram of 8279

□ 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 as shown in Figure 7.
□ 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.

Block diagram of 8279:

The functional block diagram of 8279 is as shown in Figure 8.

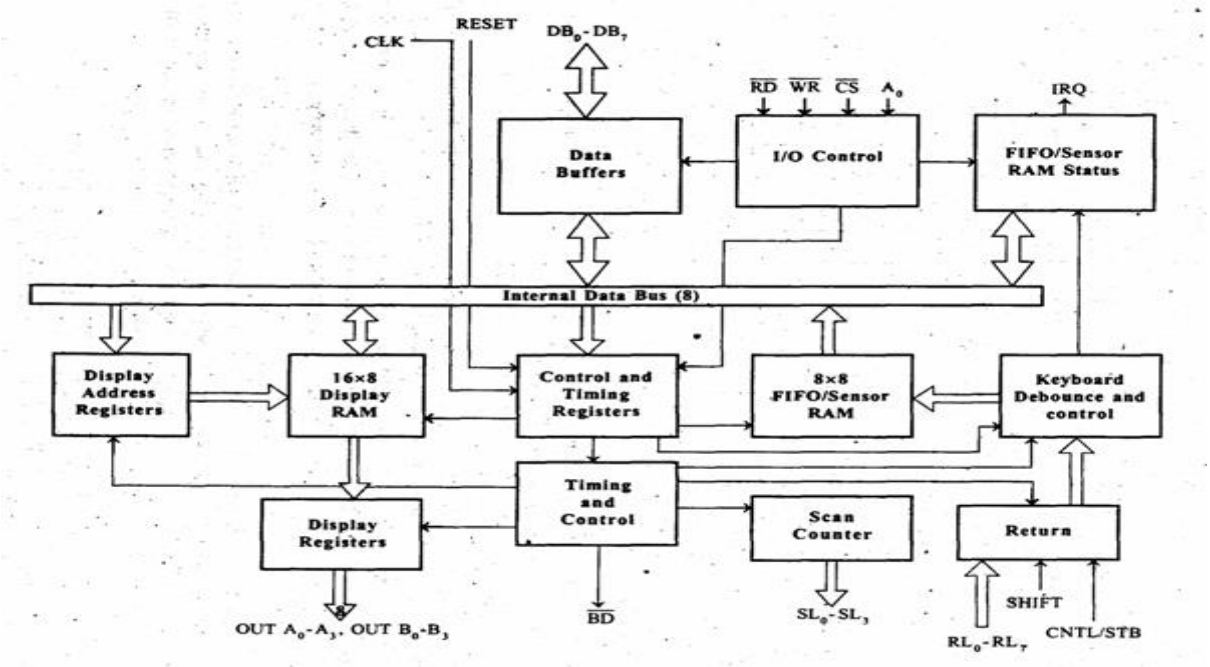
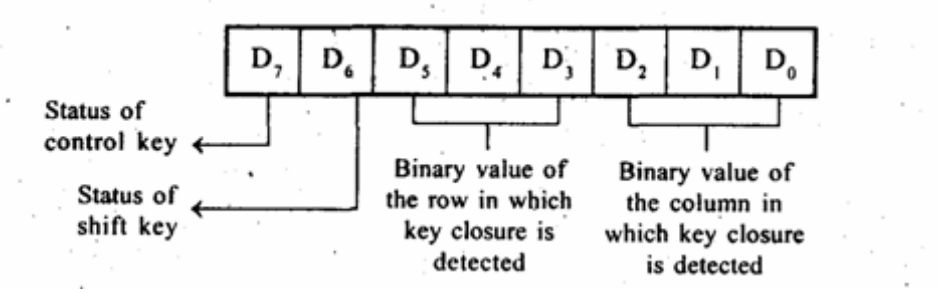


Figure 4.2:Block diagram of 8279

- The four major sections of 8279 are keyboard, scan, display and CPU interface.

Keyboard Section:

- The keyboard section consists of eight return lines RL0 - RL7 that can be used to form the columns of a keyboard matrix.
- It has two additional input : shift and control/strobe. The keys are automatically debounced.
- The two operating modes of keyboard section are 2-key lockout and N-key rollover.
- In the 2-key lockout mode, if two keys are pressed simultaneously, only the first key is recognized.
- In the N-key rollover mode simultaneous keys are recognized and their codes are stored in FIFO.
- The keyboard section also have an 8 x 8 FIFO (First In First Out) RAM.
- The FIFO can store eight key codes in the scan keyboard mode. The status of the shift key and control key are also stored along with key code. The 8279 generate an interrupt signal when there is an entry in FIFO. The format of key code entry in FIFO for scan keyboard mode is,



- In sensor matrix mode the condition (i.e., open/close status) of 64 switches is stored in FIFO RAM. If the condition of any of the switches changes then the 8279 asserts IRQ as high to interrupt the processor.

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.

Programming the 8279:

- The 8279 can be programmed to perform various functions through eight command words.

II. Interfacing of 8279 with 8085

In a microprocessor b system, when keyboard and 7-segment LED display is interfaced using ports or latches then the processor has to carry the following task.

- Keyboard scanning

- Key debouncing
- Key code generation
- Sending display code to LED
- Display refreshing

Interfacing 8279 with 8085 processor:

- A typical Hexa keyboard and 7-segment LED display interfacing circuit using 8279 is shown in figure 9.

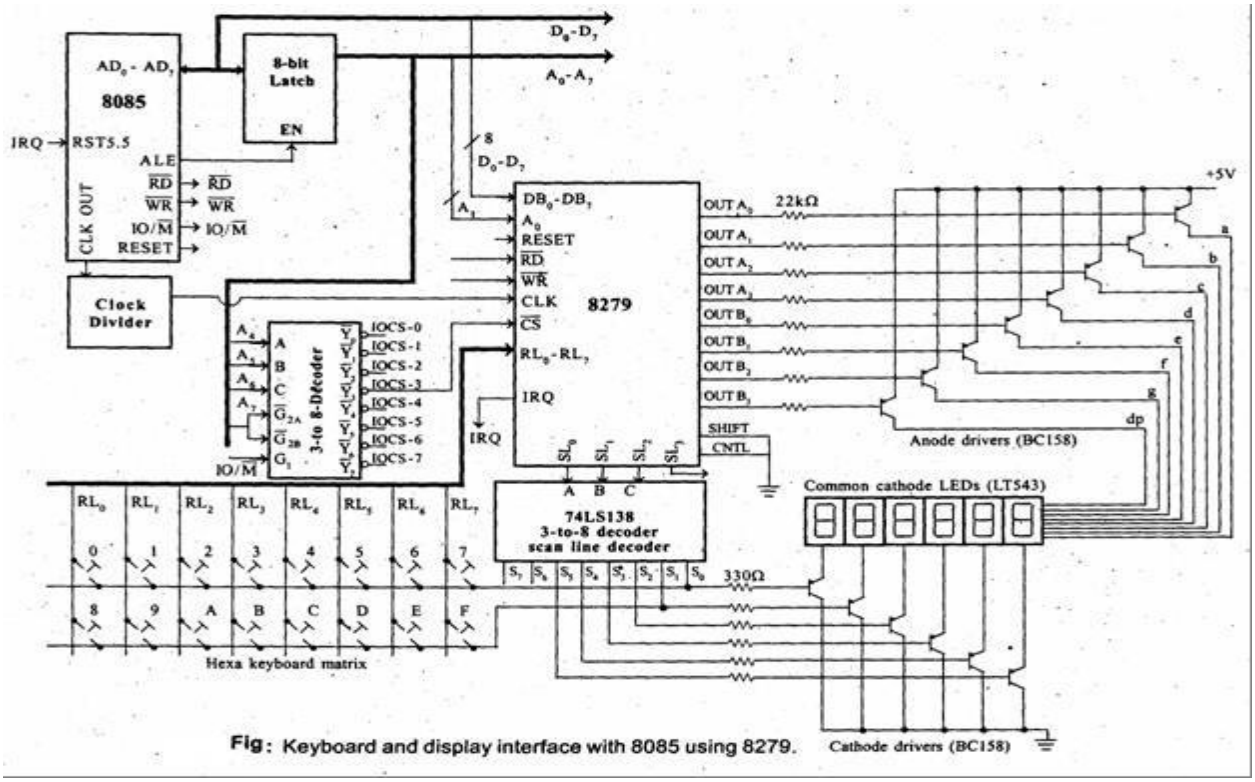


Fig 4.3 keyboard and display interface with 8085 using 8279

- The circuit can be used in 8085 microprocessor system and consist of 16 numbers of hexa-keys and 6 numbers of 7-segment LEDs.

- The 7-segment LEDs can be used to display six digit alphanumeric character.

- The 8279 can be either memory mapped or I/O mapped in the system. In the circuit shown is the 8279 is I/O mapped.

- The address line A0 of the system is used as A0 of 8279.

- The clock signal for 8279 is obtained by dividing the output clock signal of 8085 by a clock divider circuit.

- The chip select signal is obtained from the I/O address decoder of the 8085 system. The chip select signals for I/O mapped devices are generated by using a 3-to-8 decoder.

- The address lines A4, A5 and A6 are used as input to decoder. The address line A7 and the control signal IO/M (low) are used as enable for decoder.

- The chip select signal IOCS-3 is used to select 8279.

- The I/O address of the internal devices of 8279 are shown in table.

Internal Device	Binary Address								Hexa Address
	Decoder input and enable				Input to address line of 8279				
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Data register	0	0	1	1	x	x	x	0	30
Control register	0	0	1	1	x	x	x	1	31

- The circuit has 6 numbers of 7-segment LEDs and so the 8279 has to be programmed in encoded scan. (Because in decoded scan, only 4 numbers of 7-segment LEDs can be interfaced)

- In encoded scan the output of scan lines will be binary count. Therefore an external, 3-to-8 decoder is used to decode the scan lines SL0, SL1 and SL2 of 8279 to produce eight scan lines S0 to S7.

- The decoded scan lines S0 and S1 are common for keyboard and display.

- The decoded scan lines S2 to S5 are used only for display and the decoded scan lines S6 and S7 are not used in the system.

- Anode and Cathode drivers are provided to take care of the current requirement of LEDs.

- The pnp transistors, BC 158 are used as driver transistors.

- The anode drivers are called segment drivers and cathode drivers are called digit drivers.

- The 8279 output the display code for one digit through its output lines (OUT A0 to OUT A3 and OUT B0 to OUT B3) and send a scan code through, SL0- SL3.

- The display code is inverted by segment drivers and sent to segment bus.

- The scan code is decoded by the decoder and turns ON the corresponding digit driver. Now one digit of the display character is displayed. After a small interval (10 millisecond, typical), the display is turned OFF (i.e., display is blanked) and the above process is repeated for next digit. Thus multiplexed display is performed by 8279.

- The keyboard matrix is- formed using the return lines, RL0 to RL3 of 8279 as columns and decoded scan lines S0 and S1 as rows.

- A hexa key is placed at the crossing point of each row and column. A key press will short the row and column. Normally the column and row line will be high.

- During scanning the 8279 will output binary count on SL0 to SL3, which isdecoded by decoder to make a row as zero. When a row is zero the 8279 reads the columns. If there is a key press then the corresponding column will be zero.

- If 8279 detects a key press then it wait for debounce time and again read the columns to generate key code.

- In encoded scan keyboard mode, the 8279 stores an 8-bit code for each valid key press. The keycode consist of the binary value of the column and row in which the key is found and the status of shift and control key.

- After a scan time, the next row is made zero and the above process is repeated and so on. Thus 8279 continuously scan the keyboard.

DAC INTERFACING WITH 8085

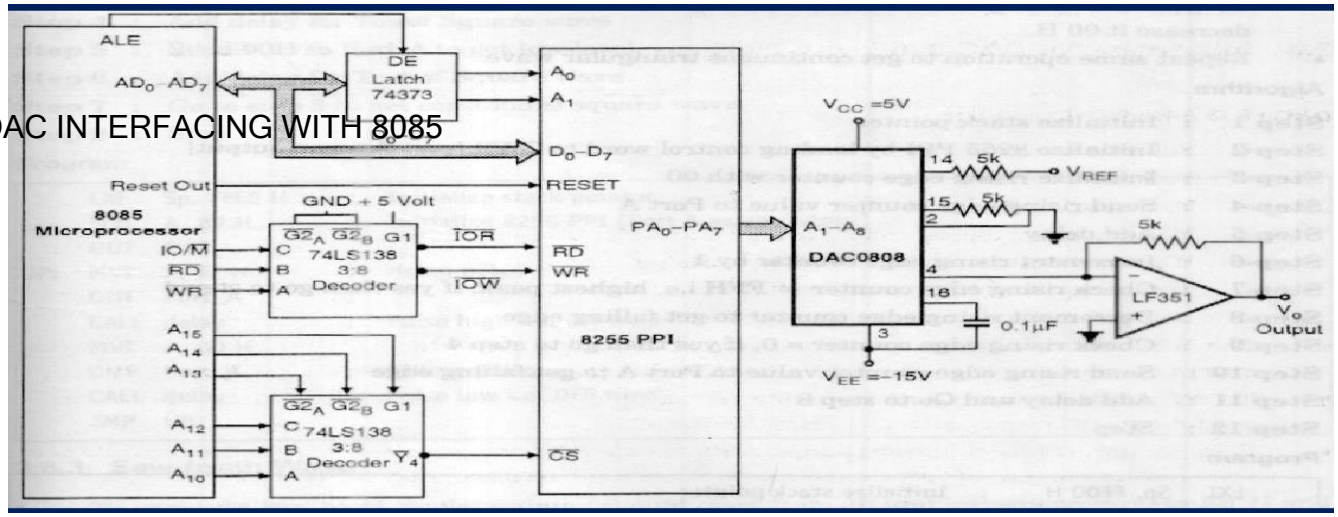


Fig 4.4 DAC interfacing

The DAC0800 can be interfaced to 8085 system bus by using an 8-bit latch and the latch can be enabled by using one of the chip select signal generated for I/O devices. A simple schematic for interfacing is DAC0800 with 8085 is shown in the fig. The DAC0800 can be interfaced to 8085 system bus by using an 8-bit latch and the latch can be enabled by using one of the chip select signal generated for I/O devices. A simple schematic for interfacing

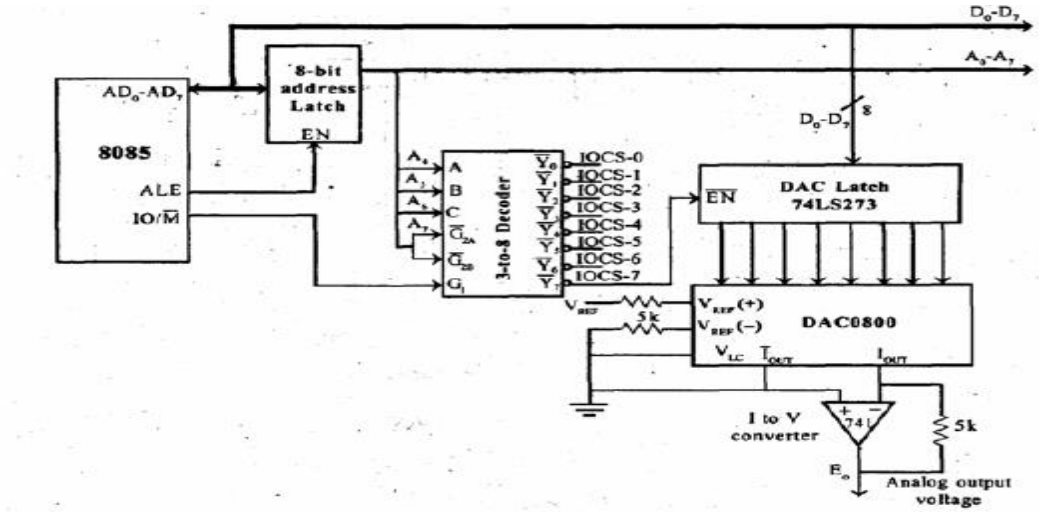


Fig 4.5 : DAC Interface

In this schematic the DAC0800 is interfaced using an 8-bit latch 74LS273 to the system bus. The 3-to-8 decoder 74LS 138 is used to generate chip select signals for I/O devices. The address lines A4, A5 and A6 are used as input to decoder.The address line A7 and the control signal IO/M (low) are usedas enable for decoder. The decoder will generate eight chip select signals and in thisthe signal IOCS-7 is used as enable for latch of DAC.

The I/O address of the DAC is shown in table.

Device	Binary Address								Hexa Address
	Decoder Input and enable				Unused address lines				
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
DAC Latch 74LS273	0	1	1	1	x	x	x	x	70

- In order to convert a digital data to analog value, the processor has to load the data to latch.
- The latch will hold the previous data until next data is loaded.
- The DAC will take definite time to convert the data. The software should take care of loading successive data only after the conversion time. The DAC 0800 produces a current output, which is converted to voltage output using I to V converter.
- 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. Every analog output will be a multiple of the resolution,

ADC INTERFACING WITH 8085

The ADC 0808 is 8-channel 8-bit ADC chip. It has 8 analog inputs i.e. IN0-IN7.One of these channels is selected by sending address to a address line of ADC. The logic level and selected channel is as shown:

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

The analog signal is connected to channel 3.The digital equivalent data D0-D7 is connected to PA0-PA7 of Port A. The PC0, PC1 and PC2 lines of Port C are connected to channel select address lines of 8255.

PC3 is connected to SOC (Start of conversion) and ALE signal (Input signal). EOC (End of conversion) which is an output signal of 8255 connected to PC7 of Port C. The PB0 of Port B is connected to OE (Output Enable) input signal of ADC.

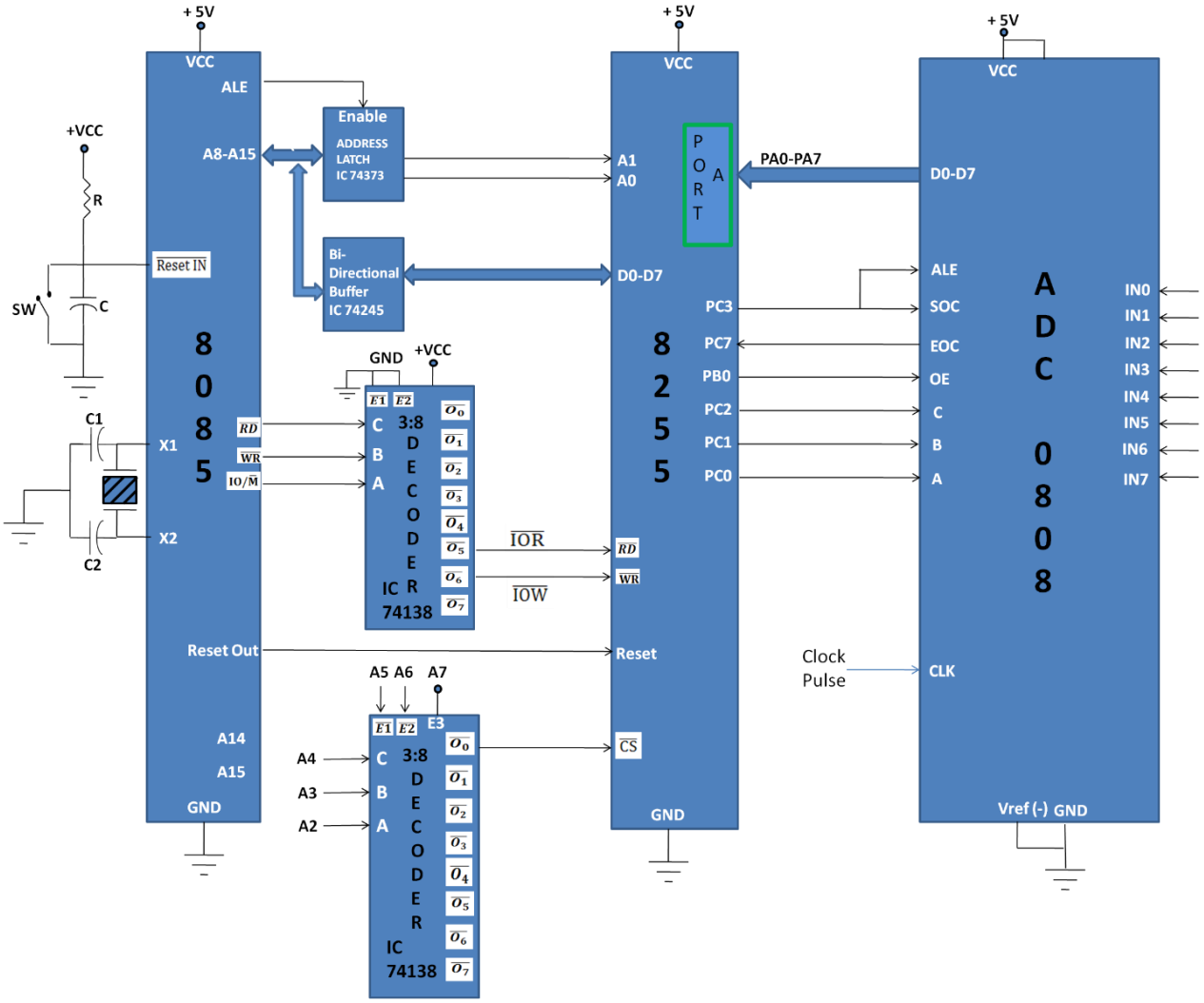


Fig 4.6 ADC INTERFACING WITH 8085

SEVEN SEGMANT DISPLAY

Interface the 8085 Microprocessor System with seven segment display through its programmable I/O port 8255. Seven segment displays (as shown in Figure 1) is often used in the digital electronic equipments to display information regarding certain process.

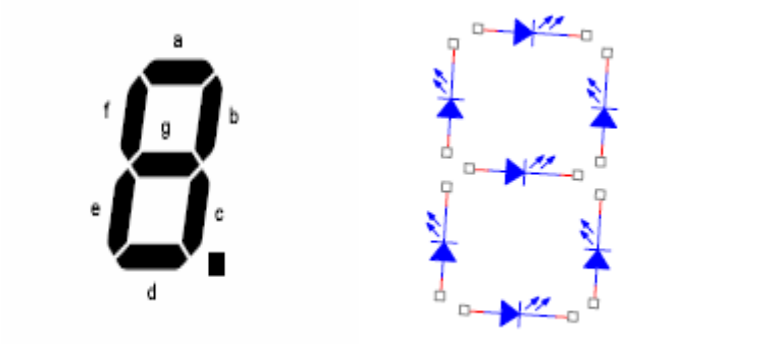


Fig 4.7 Seven Segment Display

There are two types of seven segment display; common anode and common cathode. The differences between these two displays are shown in Figure 2a and 2b. The internal structure of the seven segment display consist of a group of Light Emitting Diode (LED)

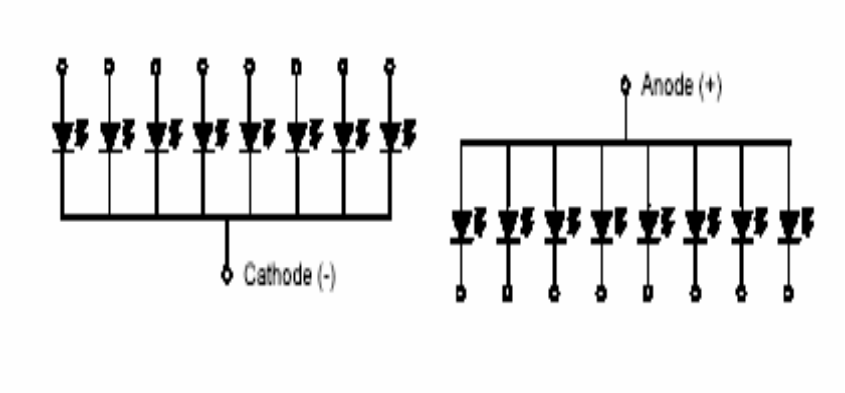


Figure 4.8 - Common Cathode and Common Anode

For common cathode, the segment will light up when logic '1' (+V) is supplied and it will light off when logic '0' (OV) is supplied. While for common anode, logic '1' will light off the segment and logic '0' will light up the segment. Therefore to display number '0' on the seven segment display, segment a, b, c, d, e and f must light up. For common cathode, logic '1' should be given to the related segment whereas in the case of common anode, logic '0' should be given to the necessary segment.

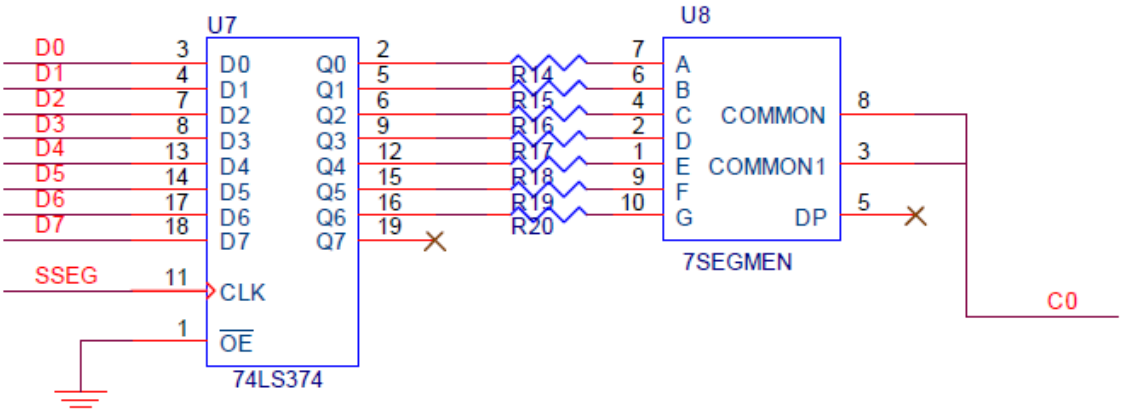


Figure 4.9 common anode

Comparison between 8085 and Z80 Microprocessors

S.No.	8085 Microprocessor	Z80 Microprocessor
1	Data Lines are MULTIPLEXED	It has no MULTIPLEXED lines
2	74 instructions	158 Instructions
3	Operates at 3 to 5MHz	Operates at 4 to 20 MHz
4	It has 5 interrupts	It has two interrupts
5	No on board dynamic memory	It has on board logic to refresh Dynamic memory
6	It contains no Index register	It has two Index register
7	It contains SIM & RIM	It contains no SIM & RIM

Comparison between 8085 and MC6800 Microprocessors

S.No.	8085 Microprocessor	MC6800 Microprocessor
1	It operates on Clock frequency of 3 to 5 MHz.	It operates at 1 MHz frequency.
2	8085 has no Index register.	It has one index register.
3	8085 has on board clock logic circuit.	No clock logic circuit.
4	8085 has one Accumulator	MC6800 has two Accumulator

	Register.	Registers.
5	8085 has five interrupts.	MC 6800 have two interrupts.
6	It has total 674 Instructions.	MC6800 has total 72 instructions.

Alphanumeric LCD display:

Liquid Crystal displays are created by sandwiching a thin 10-12 μm layer of a liquid-crystal fluid between two glass plates. A transparent, electrically conductive film or backplane is put on the rear glass sheet. Transparent sections of conductive film in the shape of the desired characters are coated on the front glass plate.

When a voltage is applied between a segment and thebackplane, an electric field is created in the region under the segment. This electric field changes the transmission of light through the region under the segment film.

There are two commonly available types of LCD:

- Dynamic scattering and
- Field-effect.

The Dynamic scattering types of LCD: It scrambles the molecules where the field is present. This produces an etched-glass-looking light character on a dark background Field-effect types use polarization to absorb light where the electric field is present. This produces dark characters on a silver- gray background.

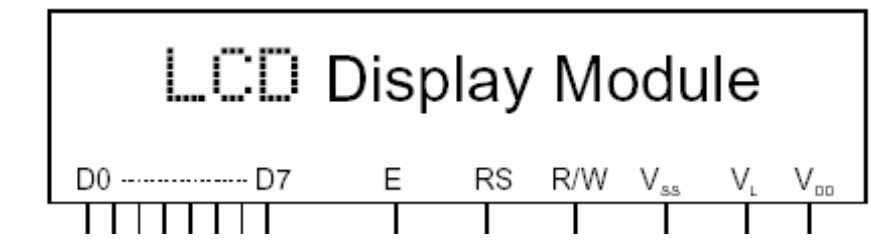


Figure 4.10 Pin Assignment

The pin assignment shown below is an industry standard for small (80 characters or less) alphanumeric LCD - modules.

Pin number Symbol I/O Function

1- Vss - Power supply (GND)

2 -VDD - Power supply (+5V)

3 -Vee - Contrast adjust

4 -RS I 0 = Command input/output 1 = Data input/output

5 -R/W I 0 = Write to LCD module 1 = Read from LCD module

6 -E I Enable signal (Data strobe)

7 -DB0 I/O Data bus line 0 (LSB)

8 -DB1 I/O Data bus line 1

9 -DB2 I/O Data bus line 2

10- DB3 I/O Data bus line 3

11 -DB4 I/O Data bus line 4

12 -DB5 I/O Data bus line 5

13 -DB6 I/O Data bus line 6

14 -DB7 I/O Data bus line 7 (MSB)

The LCD module requires 3 control lines and either 4 or 8 I/O lines for the data bus. The user may select whether the LCD is to operate with a 4-bit data bus or an 8-bit data bus. If a 4-bit data bus is used, the LCD will require a total of 7 data lines (3 control lines plus the 4 lines for the data bus). If an 8-bit data bus is used, the LCD will require a total of 11 data lines (3 control lines plus the 8 lines for the data bus). The three control lines are referred to as E, RS, and R/W. The E line is called "Enable." This control line is

used to tell the LCD that you are sending it data. To send data to the LCD, your program should first set this line high (1) and then set the other two control lines (RS & RW) and put data on the data bus (DB0- DB8). When the other lines are completely ready, bring E low (0) again.

The 1 to 0transition tells the LCD to take the data currently found on the other control lines and on the data bus and to treat it as a command. The RS line is the "Register Select" line. When RS is low (0), the data is to be treated as a command or special instruction (such as clear screen, position cursor, etc.). When RS is high (1), the data being sent is text data which should be displayed on the LCD screen. For example, to display the letter "T" on the screen you would set RS high. The RW line is the "Read/Write" control line. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (orreading) the LCD. Finally, the data bus consists of 4 or 8 lines (depending on the mode of operation selected by the user). In the case of an 8-bit data bus, the lines are referredto as DB0, DB1, DB2, DB3, DB4, DB5, DB6, and DB7

• Most LCD’s require a voltage of 2 or 3 V between the backplane and a segment to turn on the segment.

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

SCHOOL OF ELECTRICAL AND ELECTRONICS
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

UNIT 5
Microprocessor , Interfacing and Its Applications-SECA1508

STEPPER MOTOR INTERFACING USING 8085:

Stepper motor is an electromechanical device that rotates through fixed angular steps when digital inputs are applied. It is suitable for precise position, speed and direction control which are required in automation system.

The angle through which stepper motor rotates with a fixed angle for each digital data is called step angle.

Different stepper motor has different step angle. The more frequently used steppermotor has step angle of 0.9 degrees and 1.8 degrees.

Depending on the sequence applied to stepper motor, it can be classified in two category:

1. 4- Step sequence or full step sequence

2. 8- Step sequence or half step sequence

4- Step sequence or full step sequence:

4- Step sequence binary pattern	HEX code	Comments			
A	B	C	D		
1	0	1	0	0AH	Sequence for Clock wise rotation
1	0	0	1	09H	
0	1	0	1	05H	
0	1	1	0	06H	
0	1	1	0	06H	Sequence for anti-clockwise rotation
0	1	0	1	05H	
1	0	0	1	09H	
1	0	1	0	0AH	

Chips select Logic:

Chip select address lines	Address lines to select port	HEX address	Selected I/O						
A7	A6	A5	A4	A3	A2	A1	A0		
1	0	0	0	0	0	0	0	80H	PORT A
1	0	0	0	0	0	0	1	81H	PORT B
1	0	0	0	0	0	1	0	82H	PORT C
1	0	0	0	0	0	1	1	83H	Chip select register

The program in look up table if the 4-step sequence for clock wise then stepper motor will rotate in clockwise direction and if the 4-step sequence for anti-clock wise then stepper motor will rotate in anti-clockwise direction. Speed control of stepper motor is achieved by writing program to rotate stepper motor continuously in delay program. We can change the delay between two steps and thus change the speed of stepper motor.

Program:

LABEL	OPCODE	OPERAND	COMMENT
	LXI	SP,2800H	Initialize Stack pointer
	MVI	A, 80H	Initialize 8255
	OUT	83H (CWR)	
	MVI	B, 32H	Initialize repeated count
REPEAT:	LXI	H, 2100H	Initialize 4-step sequence
	MVI	C,04H	Initialize 4-step sequence from look up table
BACK:	MOV	A,M	
	OUT	80H (PORT A)	Sends data to Port A
	CALL	DELAY	Provide time interval between steps
	INX	H	Increment look up table
	DCR	C	Decrement 4-step count
	JNZ	BACK	Is count= "00"? if no then jump to BACK
	DCR	B	Is count= "00"? if yes then decrement repeated count
	JNZ	REPEAT	Repeated count is repeated for further rotation
	HLT		

In the above program in look up table if the 4-step sequence for clock wise then stepper motor will rotate in clockwise direction and if the 4-step sequence for anti-clock wise then stepper motor will rotate in anti-clockwise direction. Speed control of stepper motor is achieved by writing program to rotate stepper motor continuously in delay program. We can change the delay between two steps and thus change the speed of stepper motor.

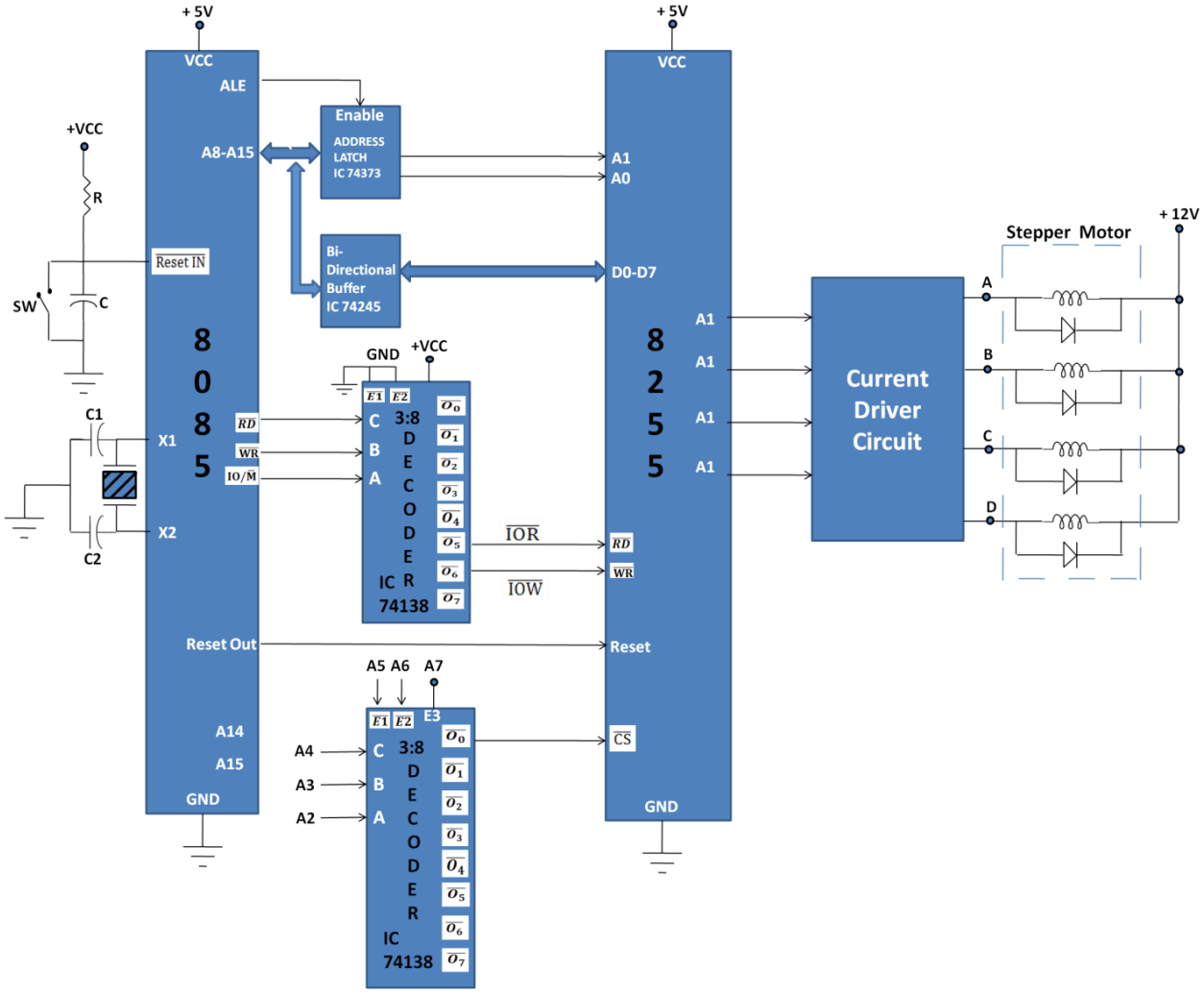


Fig 5.1 Stepper motor interface using 8085

Temperature controller using 8085:

Temperature measurement is used in variety of applications like furnace, water bath, oven, etc. with the help of transducers like thermocouple. The output of thermocouple is proportional to temperature which is in milliVolts. Therefore to drive further stages of system, this signal is amplified using instrumentation amplifier. The amplified output is fed to channel 3 of ADC and 8085 provides High to Low SOC and ALE signal. When ADC completes the conversion, 8085 reads the equivalent digital data from Port A which is the current value of temperature of object. This value of measured temperature is then sent to display system.

For measuring temperature of furnace, water bath, etc. 8085 1st measures current temperature (t1) and compares with the reference temperature (T1) at which the temperature is to be kept constant.

If the measure temperature (t1) is greater than reference temperature (T1) then 8085 sends control signal to the transistorized relay circuit through Port B and turns OFF the heating process to maintain temperature at desired level.

If the measure temperature (t1) is less than reference temperature (T1) then 8085 sends control signal to the transistorized relay circuit through Port B and turns ON the heating process to maintain temperature at desired level, thus maintaining the temperature of furnace, bath tub, etc.

Chips select Logic:

Chip select address lines	Address lines to select port	HEX address	Selected I/O						
A7	A6	A5	A4	A3	A2	A1	A0		
1	0	0	0	0	0	0	0	80H	PORT A
1	0	0	0	0	0	0	1	81H	PORT B
1	0	0	0	0	0	1	0	82H	PORT C
1	0	0	0	0	0	1	1	83H	Chip select register

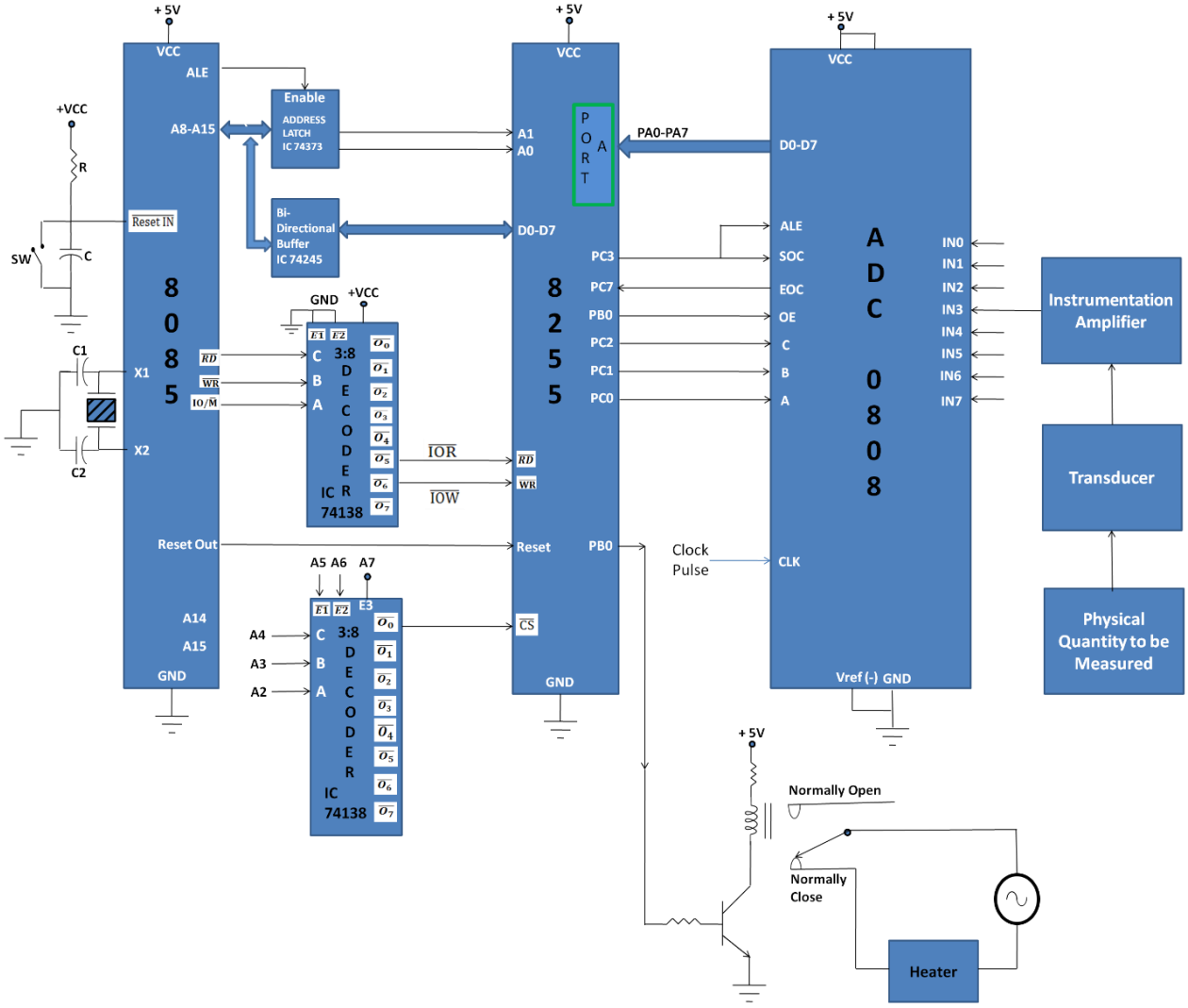


Fig 5.2 Temperature Control using 8085

Interface AC bulb using 8085

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

I/O MAP:

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

Table 2

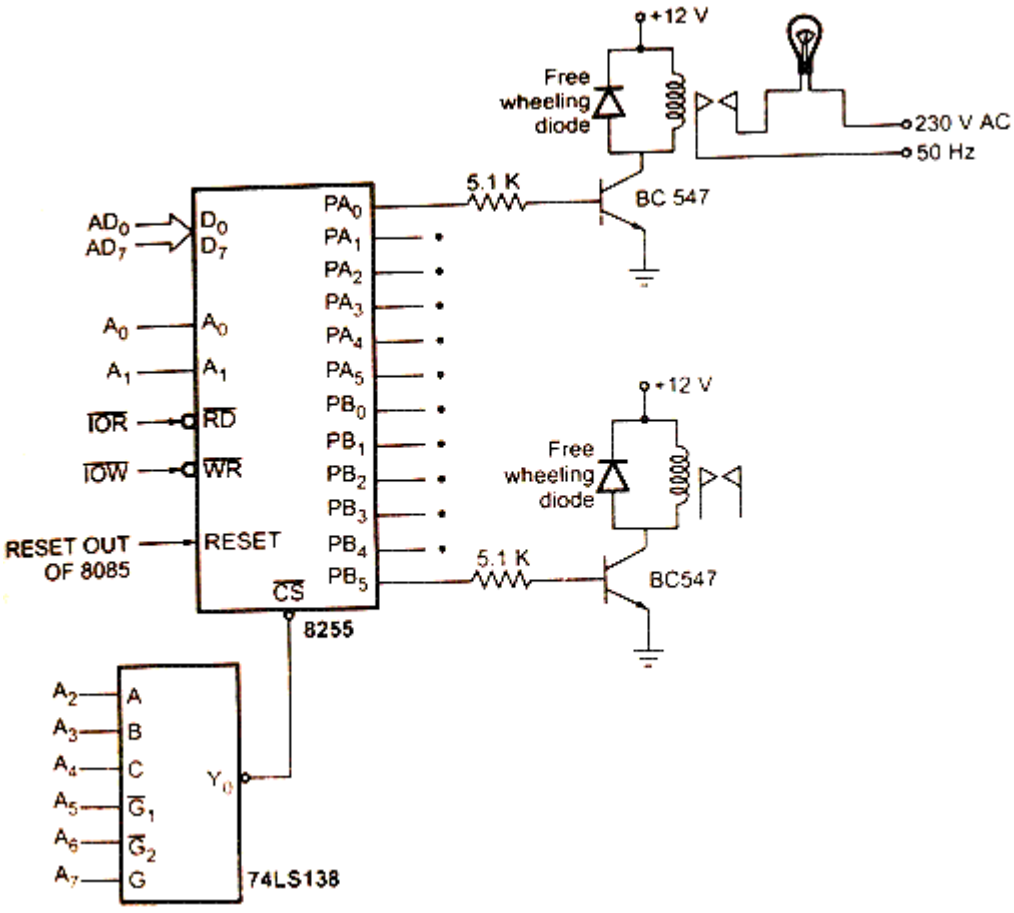


Fig 5.3 Interface AC bulb using 8085

BLOCK DIAGRAM:

TRAFFIC CONTROL SYSTEM USING 8086

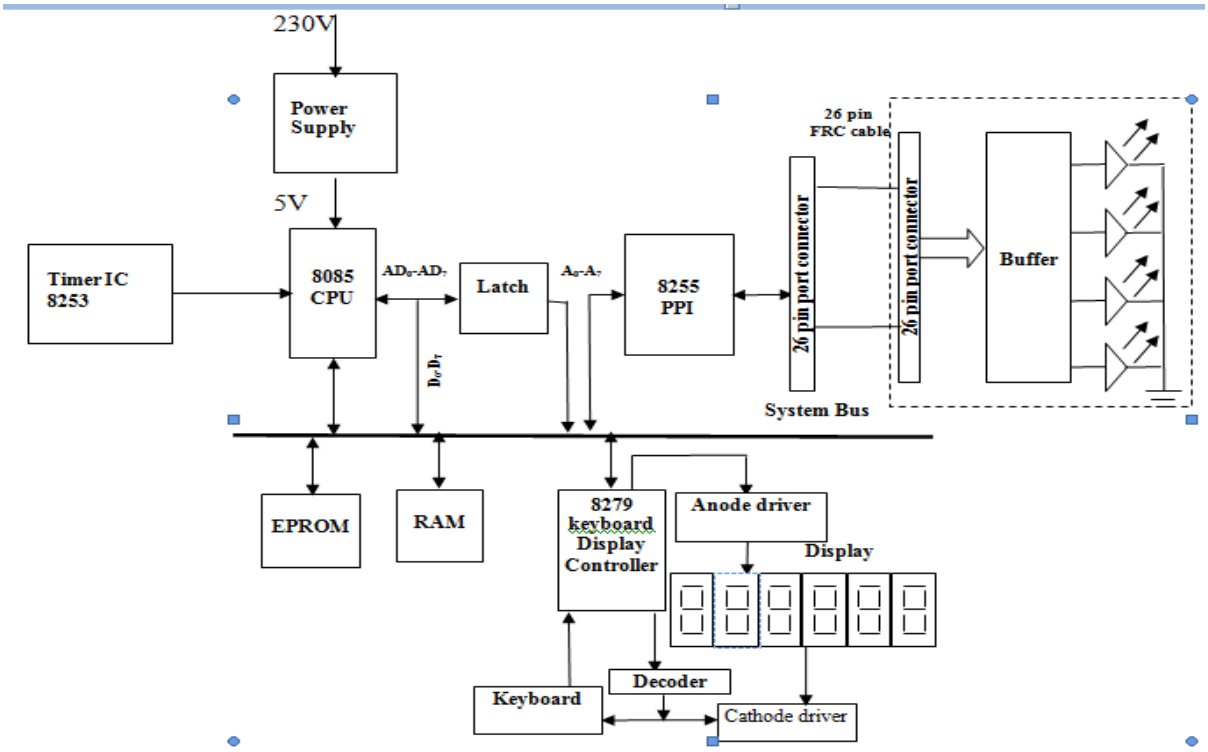


FIG 5.4 TRAFFIC CONTROL SYSTEM USING 8086

Traffic control is a big issue in today's world. Traffic jam is one of the major problems in a densely populated city like Guwahati where the population and number of running vehicles are much more than its capacity. Due to traffic jam a substantial portion of working is spent on streets which indirectly put adverse impact on economy and unavoidable road accident which results in the loss of lives. The number of vehicles is ever increasing while the city infrastructures are developing at a much slower rate. The management of traffic in City is also a tough job and only manual efforts are not sufficient to stop kind of problem. So we need machines. We need a system that can handle such a situation effectively. Today's traffic control system can handle such a situation but not that much effectively because they are static in nature. We need a system which is dynamic in nature so that it can handle traffic smoothly and such a system is called Automatic Traffic Control System