SATHYABAMA UNIVERSITY

FACULTY OF ELECTRICAL AND ELECTRONICS

SEC1201 MICROPROCESSORS AND MICROCONTROLLERS

COURSE MATERIAL

UNIT 1 INTRODUCTION TO MICROPROCESSORS

Introduction, 8085 Architecture, Pin Diagram and signals, Addressing Modes, Timing Diagram-Memory read,Memory write, I/O cycle, Interrupts and its types, Introduction to 8086 microprocessors and its operation.

MICROPROCESSORS

Microcomputer System – 8085 Architecture – 8085 Pin Diagram – Buses and Memory Operations – Addressing Modes, Basic concepts of microprocessor programming – Mnemonics – Hex code – fundamentals of assembly language - Instruction set for 8085.

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

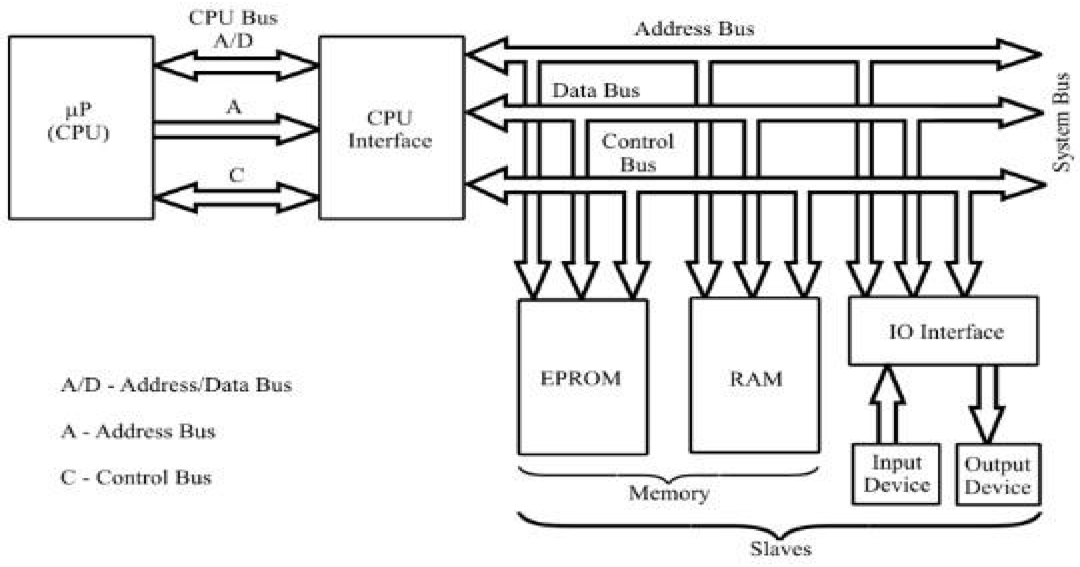


Fig.1. Microprocessor based system (organization of microcomputer)

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.

- o Transfer of data between a CPU reg. & specified memory location.
- o Performing arithmetic and logical operations on data from a specific memory location or a designated CPU register.
- o Directing the CPU to change a sequence of fetching instruction, if processing the data created a specific condition.
- o Performing housekeeping function within the CPU itself inorder to establish desired condition at certain registers.
- o It looks for control signal such as interrupts and provides appropriate responses.
- o 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

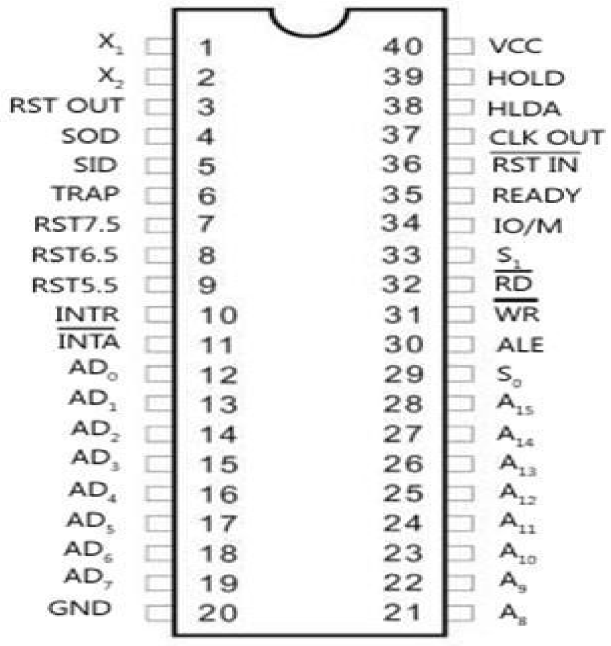


Fig .2 Pin Diagram of 8085

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.

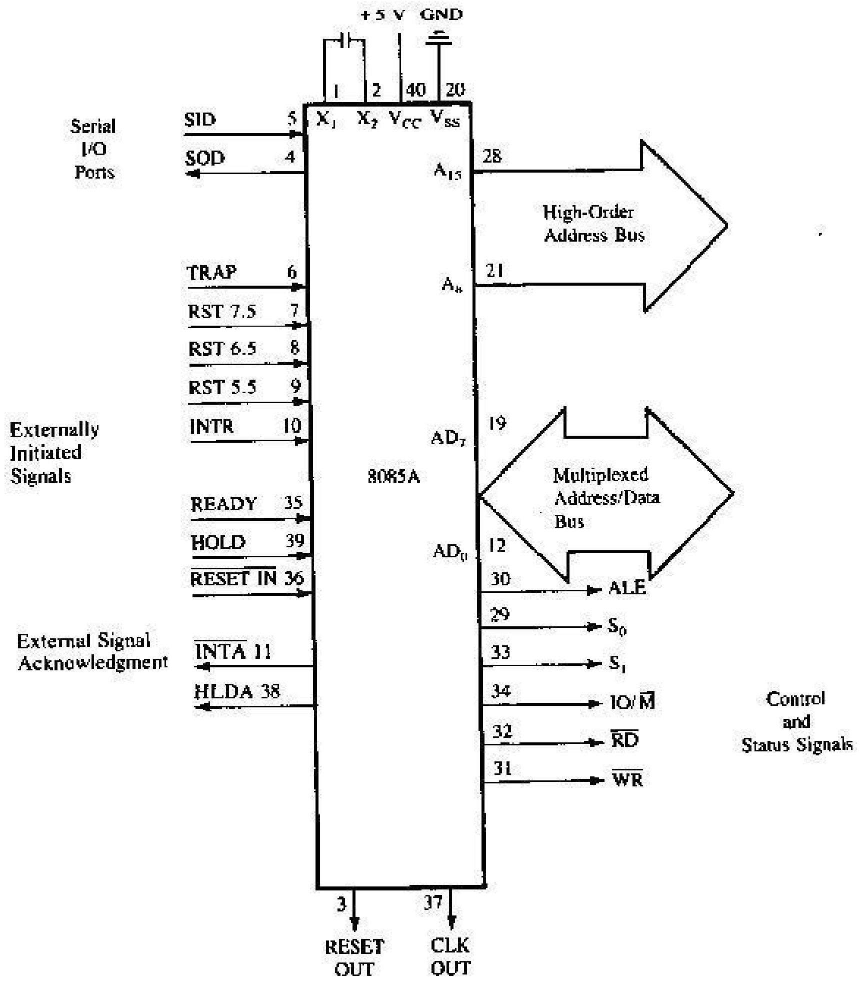


Fig: 3 Signal Classifications of 8085

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

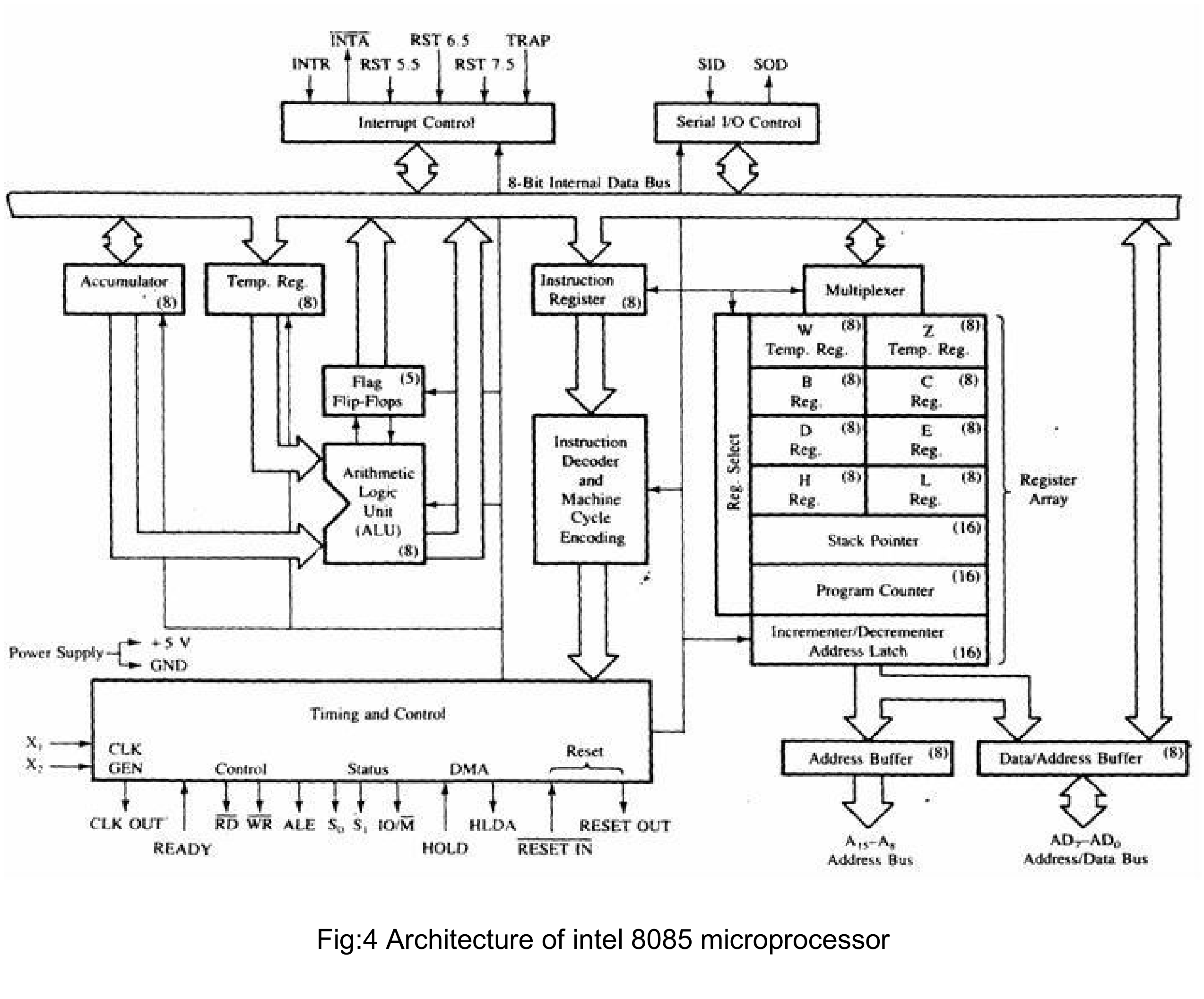


Fig:4 Architecture of intel 8085 microprocessor

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

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

Machine cycle	Status			Controls		
	$IO/\overline{M}$	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
$X \Rightarrow$ Unspecified, and $Z \Rightarrow$ 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.

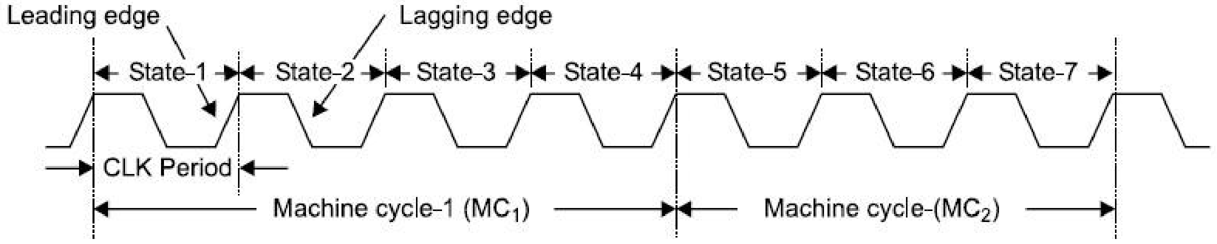


Fig.7 Machine cycle showing clock periods

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.

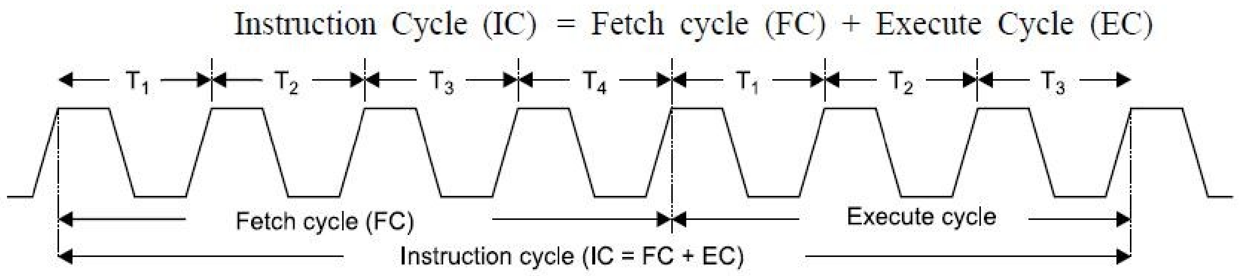

Instruction Cycle (IC) = Fetch cycle (FC) + Execute Cycle (EC)
T_1
T_2
T_3
T_4
T_1
T_2
T_3
Fetch cycle (FC)
Execute cycle
Instruction cycle (IC = FC + EC)

Fig.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.

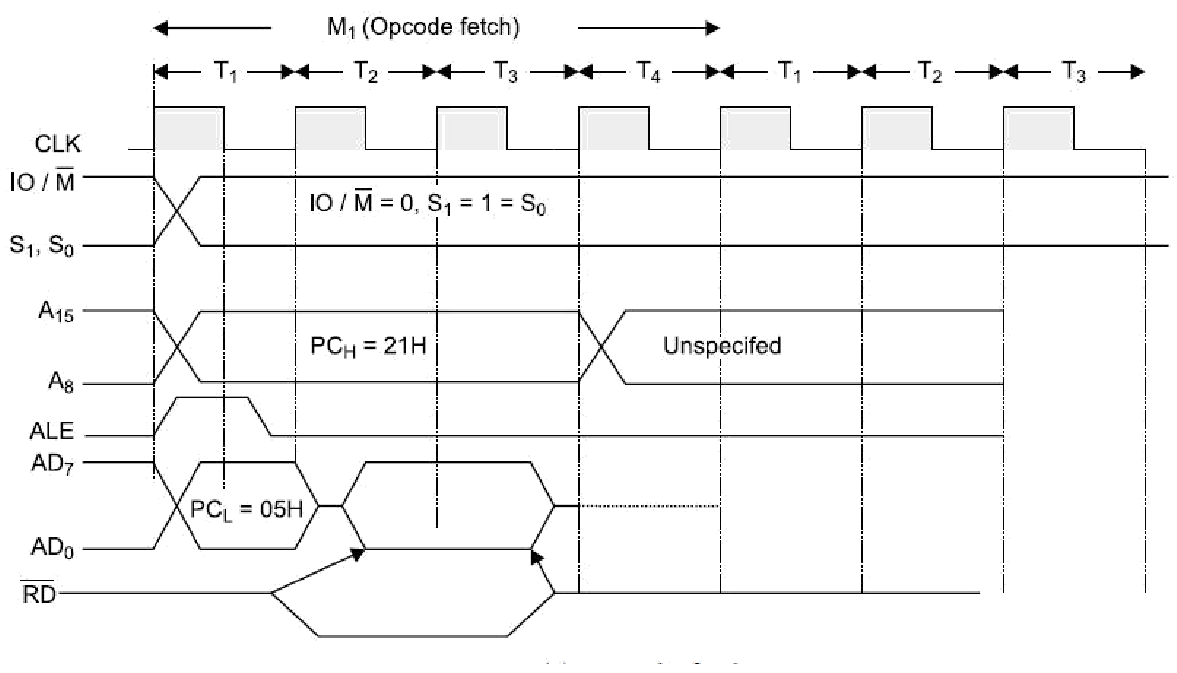


Fig. 9 Opcode fetch

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

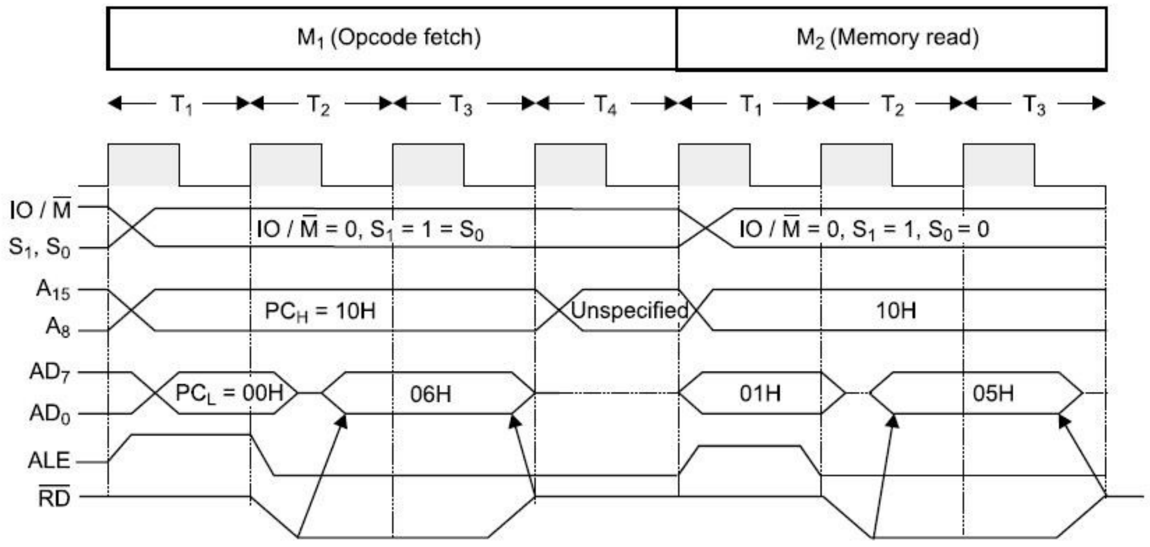


Fig. 10 Timing diagram for MVI B,05H

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.

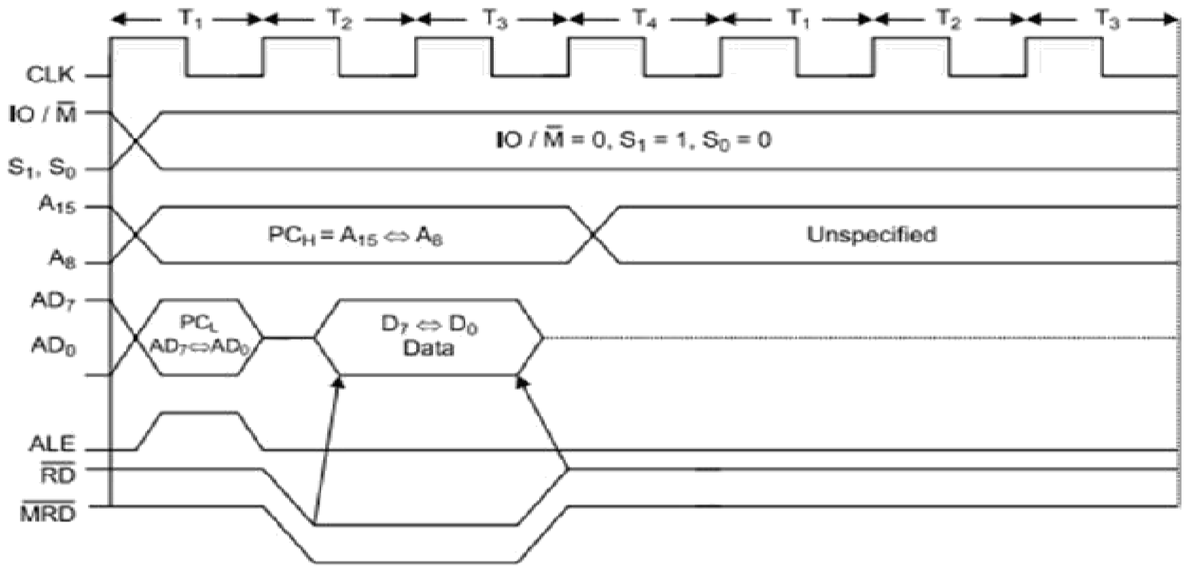


Fig 11.Memory read timing diagram

TIMING 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.

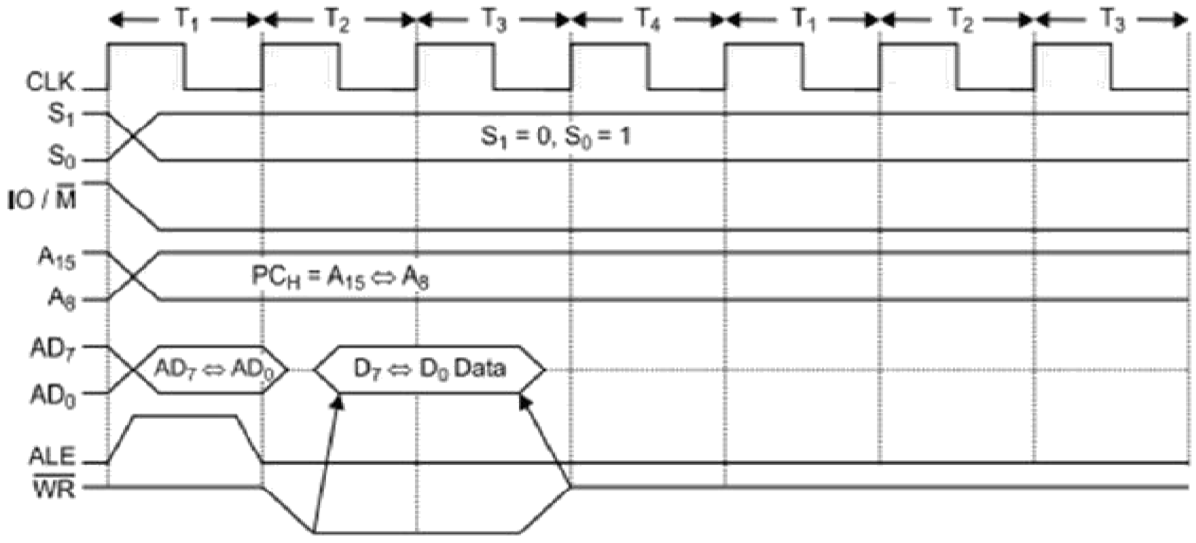


Fig 12.Memory Write timing diagram

TIMING 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.

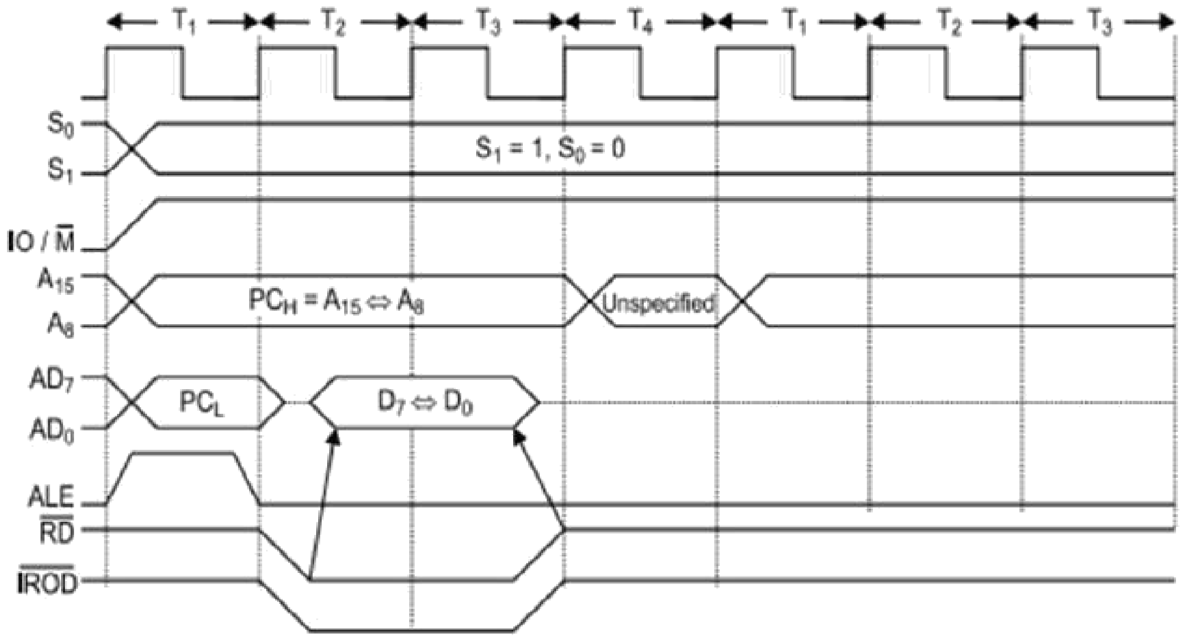


Fig 13.IO Read timing diagram

TIMING 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.

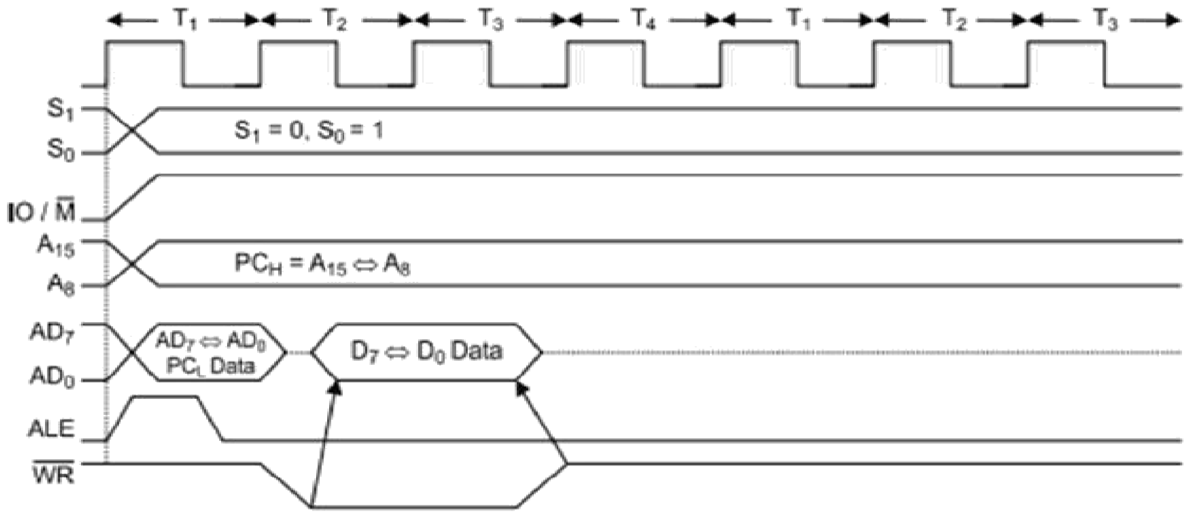


Fig 14. IO Write timing diagram

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

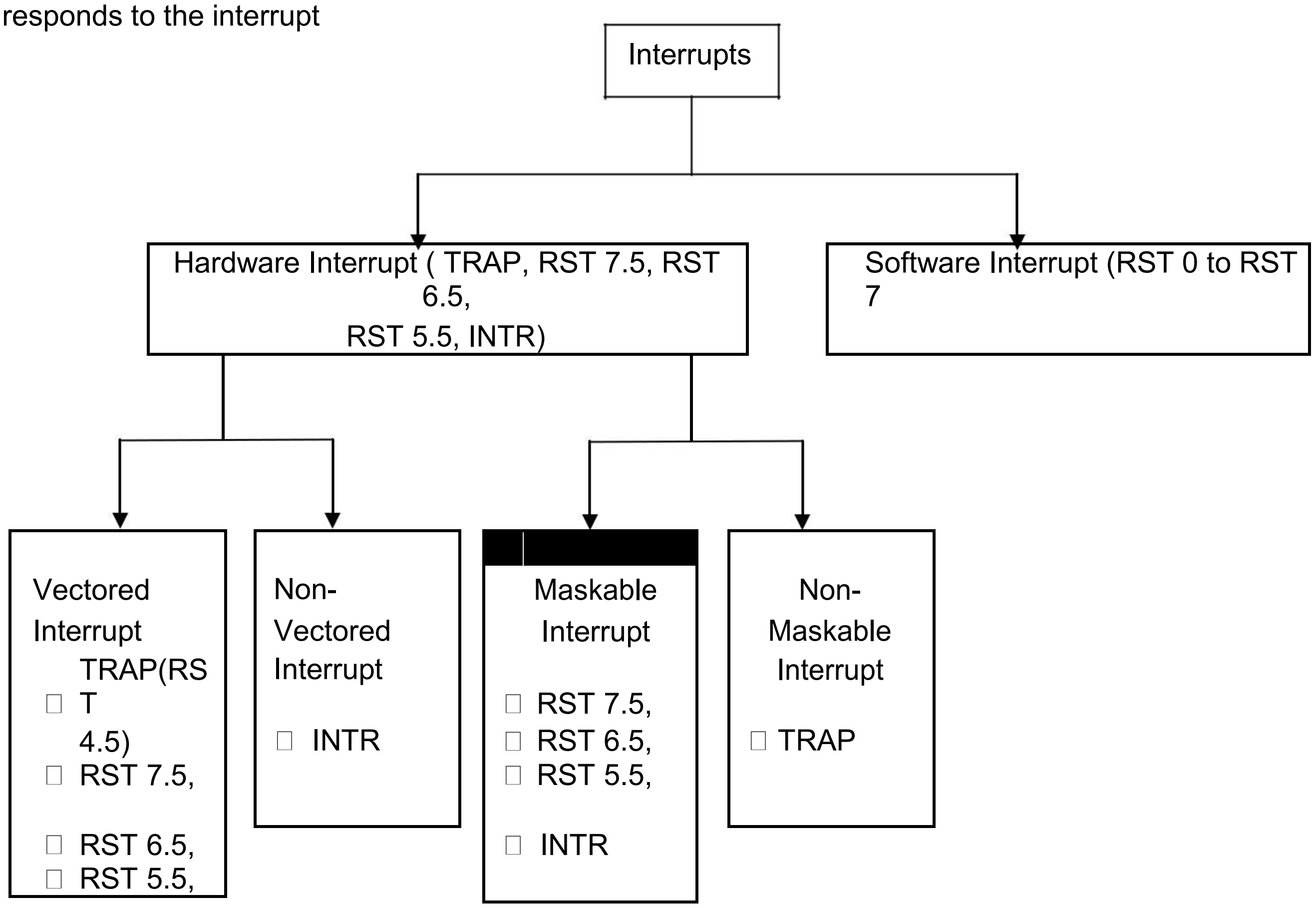
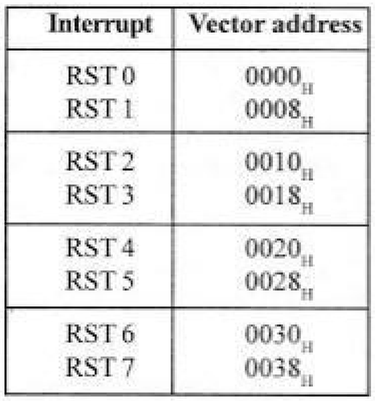


Fig 15. Structured classification of interrupts

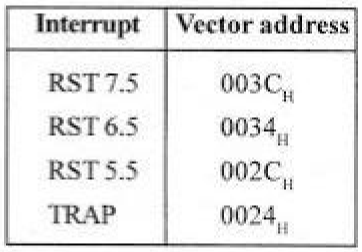
SOFTWARE INTERRUPTS OF 8085

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.

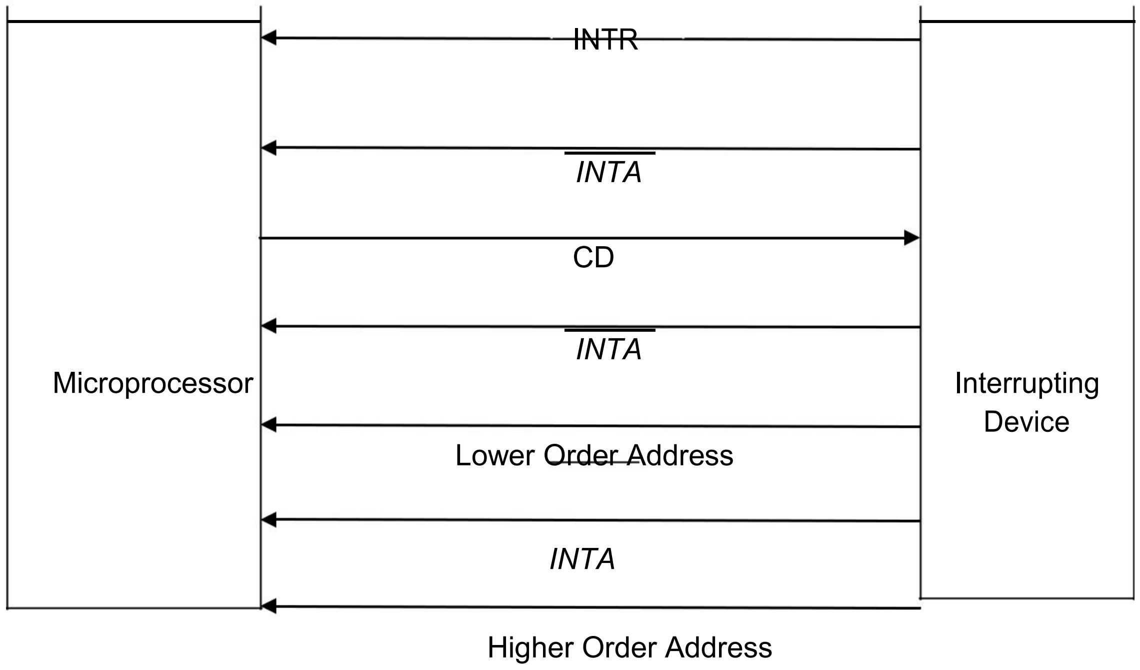


Fig 16.Work flow of Non vectored Interrupt

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)

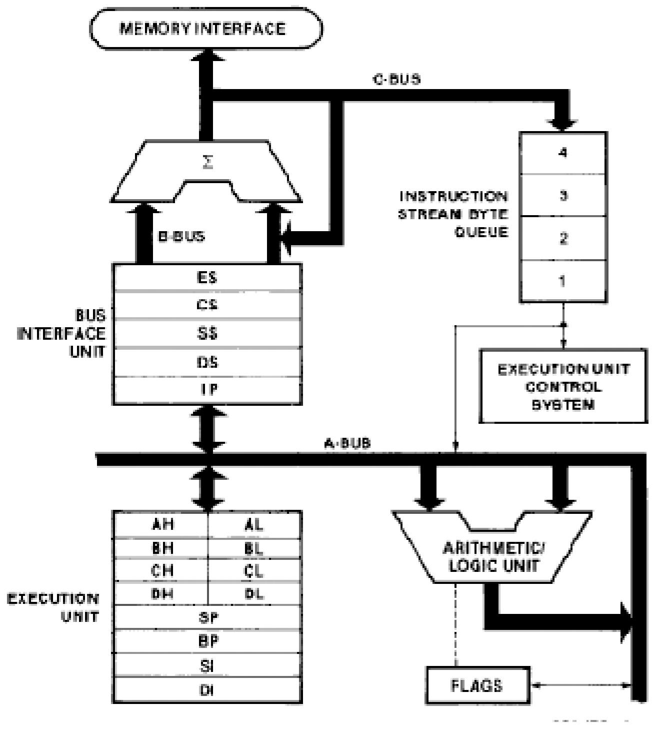
The 8085 recognizes one additional register pair called the PSW (Program Status Word). This register pair is made up of the Accumulator and the Flags registers. It is possible to push the PSW onto the stack, do whatever operations are needed, then POP it off of the stack. The result is that the contents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

INTRODUCTION TO INTEL 8086 MICROPROCESSOR

Fundamental components of 8086/8088

The 8086/8088 microprocessor consists of four functional units.

1) Execution unit (EU): decodes and executes machine instructions.
2) Arithmetic and logic unit (ALU): performs math and logical operations on command by the EU.
3) Internal storage (sometime called registers): is used for internal data storage.
4) Bus interface unit (BIU): handles all communications with the I/O via the system bus and maintains instruction queue.



Block Diagram of 8086 Microprocessor

Execution Unit (EU)

The Execution Unit controls the activity within the processor. It is responsible for retrieving binary machine-language instructions from the **Instruction Queue** maintained by the Bus Interface Unit deciphering them, and seeing to it that the correct steps are performed within the processor to carry out each instruction. If an instruction requires the use of data that is stored internally, in a register, then the Execution Unit retrieves the data from the correct register; if the instruction requires external data, from memory perhaps, then the EU requests the data from the Bus Interface Unit.

Whenever an instruction calls for an arithmetic or logical function, the EU passes the data to Arithmetic and Logic Unit together with a command telling the ALU what to do with the data. The EU then accepts the resulting data from the ALU and sees to it that it is stored into correct location (register or memory), as designated by the instruction.

Arithmetic and Logic Unit (ALU)

The ALU contains circuitry that is capable to perform arithmetic (+, -, *, /) operations and logic operations (AND, OR, NOT, XOR) operations. As the ALU completes a requested operation, it also controls individual bits of Flags register (sometimes called the program status register). It sets (1) or clear (0) the correct bits to reflect specific characteristics of the result of the operation, such as whether the result was zero or non-zero, whether it was positive or negative, or if the result is too big to be stored in a byte or word. The EU checks the status of those Flag bits whenever executing conditional jump instructions.

Bus Interface Unit (BIU)

BIU is a circuitry to response memory access and communicate with I/O devices. In most microprocessors, these functions are performed by EU, and some times the EU has to be idle due to different speed and serial execution.

In the design of the 8086/8088, Intel introduces BIU in order to avoid idle time and hold an instruction queue such that instructions can be executed in a pipeline fashion.

System Bus and Other Support Hardware Devices in IBM PC

In order to build a complete computer, there are a number of other hardware devices (chips) in the IBM PC. Some of them are:

Memory

Up to one megabyte of system memory, including that which is on the system board, the video adapter card, and any add-on memory boards.

Timer chip

Generates an external interrupt every fifty-five milliseconds to allow the PC to keep track of real time (system data and time).

8259 Interrupt Controller chip

Processes hardware (external) interrupts.

6845 CRT Controller

Located on the video adapter card. Controls the video signals to the monitor.

NEC D765 or Intel 8272 Floppy Disk Controller

Located on the disk adapter card. Acts as an interface between the processor and the disk drive.

Inter 8237 Direct Memory Access (DMA) Controller

Located on the system board. Used by the disk controller to transfer data between disk and memory.

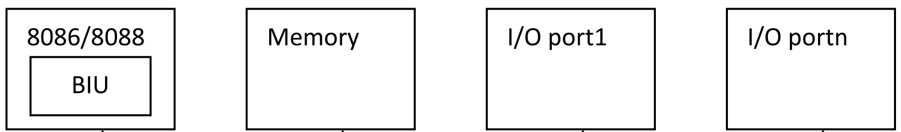
8250 Asynchronous Communications Element or Universal Asynchronous Receiver/Transmitter (UART)

Located on each serial communications adapter card (COM1, COM2, etc.).

System Bus: is a series of conductive traces of signal lines on the system board, which is used to communicate between 8086/8088 (cpu) and all other devices.

The system bus is made up of three functional parts:

1) Data bus
2) Address bus
3) Control bus



To read from memory, for example, the Bus Interface Unit puts the correct memory address onto the Address Bus and puts the command to read from memory onto the Control Bus. All devices connected to the bus see this address and command simultaneously, but only the memory-control circuitry respond to it. The memory-control circuitry is then responsible for decoding the address, retrieving the data from the appropriate memory chips, and placing the data onto the Data Bus for retrieval by the BIU.

To write to memory, the BIU puts the memory address onto the Address Bus, the byte of data onto the Data Bus, and a command to write data onto memory of the control bus. The memory-control circuitry decodes the command and the address, retrieves the data from the Data Bus, and stores it into the correct memory chips. All other circuitry simply ignores the command.

Communication with the many special-purpose microprocessors attached to the System Bus is accomplished through I/O ports. I/O ports are used for the transfer of data between the 8086/8088 processor and the other support hardware within the system. The In and OUT instructions tell the processor to input or output data through the I/O ports.

When executing an IN instruction, the BIU puts the I/O port address onto the Address Bus and puts the command to input data onto the Control Bus. The circuitry on some peripheral device attacked to the bus recognizes the red command and decodes the I/O port as the address of some register within the peripheral device. It then retrieves the data from the register and places it onto the Data Bus from which it is retrieved by the BIU and fed to the Execution Unit within the processor.

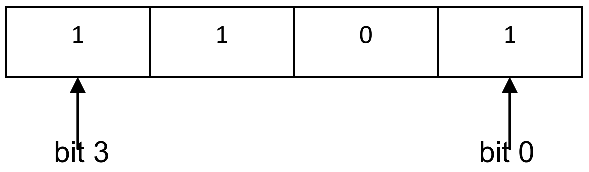
To execute an OUT instruction, the BIU puts the command to output data onto the Control Bus, and I/O address onto the Address Bus, and the data to be output onto the Data Bus. The circuitry of a peripheral device is then responsible for recognizing the output command and the I/O address and for retrieving the data from the data bus.

Memory and Internal Storage

Storage Elements

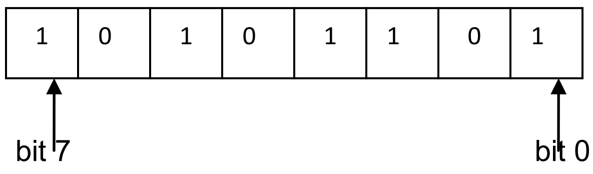
- Bits: The smallest storage element on any computer is the bit, which can have a value of 0 or 1.
- Nibbles: A nibble is a sequence of 4 bits.

e.g.



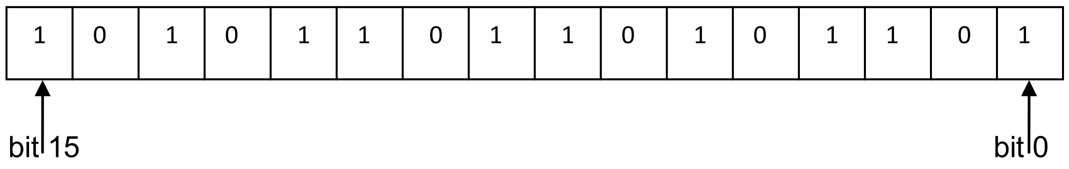
- Bytes: A byte is a sequence of 8 bits.

e.g.

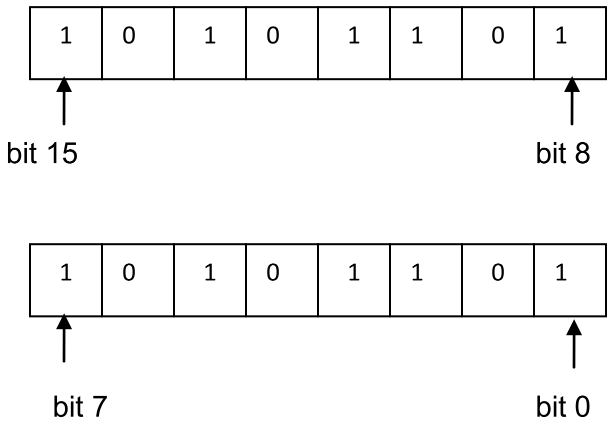


- Words: in 8086/8088, a word consists of 16 bits that can be viewed as:
  1) A horizontal sequence of 16 bits or.

e.g.



  2) A vertical sequence of 2 bytes e.g..



Note: in memory, the least significant bye (bits 0-7) is stored in lower numbered memory location, and the most significant byte (bits 8-15) is stored in the next higher numbered location.

- Double words: Two words represents a double word which is viewed as
  1) A horizontal sequence of 32 bits or
  2) A vertical sequence of 2 words or
  3) A vertical sequence of 4 bytes

  Note: in memory, its 0-7 bits would come first; bits 8-15 would be in the next higher numbered byte and so on.
- Quadword: A quadword is 2 double words, hence 4 words or 8 bytes.
- Tenbytes: A tenbyte is a sequence of 10 bytes.

Memory

Computer memory consists of an ordered sequence of storage units (8-bits called bytes), each with its own address.

- 8086/8088's memory is bytes-addressable, which means that each byte has its own address, e.g. if A is the address of a word (16 bits), then A is actually the address of the first byte (bits 0-7) of the word, A+1 is the address of the second byte (bits 8-15) of the word, and A+2 is the address of the next word.

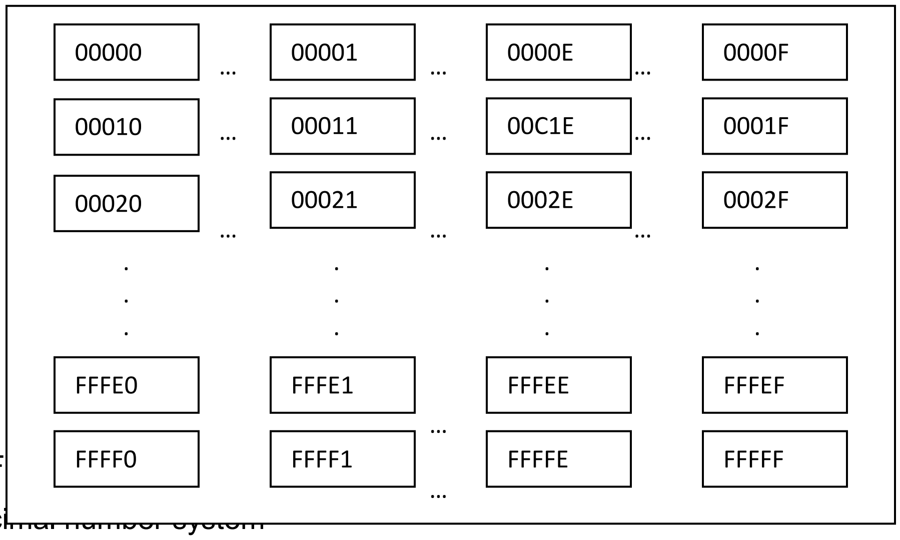
- Memory address space: (MAS): 2^{20}= 1,048,576= 1M(bytes) (address space: 0—2^{20}-1, i.e.00000-fffff: in hexadecimal)
- Map of the memory address space (MAS)

  The whole MAS is organized by a $2^{16} * 2^{4}$ matrix: there are 2^{16} paragraphs, each of which with 16 (2^{4}) bytes.

Paragraph numbers:

0000
0001
0002

FFFE
FFFF



F

- Hexadecimal number system

  The hexadecimal number system uses sixteen characters to represent umbers:0 through 9 and A through F; A is worth ten, B is worth eleven, …, and F is fifteen. In assembly program, the hexadecimal will be indicated by the letter h.

  Example: 7Fh is the number 127(7*16+15*1) and its equivalent binary number is 01111111b.

  The conversion between binary and hexadecimal is simple. To covert from hexadecimal to binary: just represent each hexadecimal digit by four binary digits: Example: 74Dh = 011101001101b

  To convert from binary to hexadecimal: break the binary number into groups of four digits, and convert each group into one hexadecimal digit:

  Example: 101001001110b = A4Fh
- Character representation

  Each character is represented by a byte or two hexadecimal digits according to ASCII code.

For example:	ASCII code	Letter
	41h	A

42h	B
.	.
.	.
5Ah	Z
61h	a
62h	b
.	.
.	.
7Ah	z
30h	0
.	.
.	.
39h	9

In assembly the ASCII representation of character can be defined by enclosing the characters in paired single (‘) or double (“) quotes.

Example: ‘A’ ‘e’ “9”

Note ‘A’ = 41h, ‘a’ = 61h and “9” = 39h

- Segment-relative addresses.
  1) There are 16 bits in a segment register. The content of the segment register represents the paragraph number of the paragraph at which it begins.
  2) All memory address are formed by segment-relative format which is computed as follows:

     Absolute address=(segment register)*16 + offset (16-bit) denoted by segment register: where the offset can be content of a register (16-bit) and a 16-bit value.

     e.g.

```
CS:            1110000000000000
CS*16:     11100000000000000000 (Beginning of code segment)
IP:            0000000000001101 (offset)
-------------------------------------------------------------------------
CS : IP = 11100000000000001101 (Actual location (absolute address) in
memory)
```

     wher ter

Program stack

A program stack is a region of memory (RAM), which is defined by the assembly program. The program stack is implemented by setting the SS (stack segment) register and by initializing the SP (stack pointer) register to the byte immediately above the top of the sack segment. Data is moved onto the stack in word- sized units by an operation called a push and retrieved from the stack in word-sized units by an operation called pop in the reversed order (last in- first out).

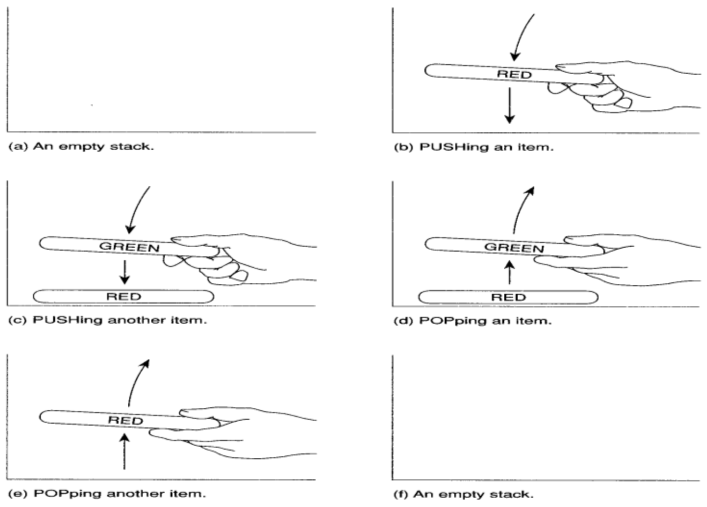


Figure: Stack operations

A Push operation decrements the SP register twice (-2) and put the word at SS : SP. A Pop operation retrieves the word at SS: SP and increments the SP register twice (+2)

Example: A program stack is to use 100h bytes and SS= 58A1h, A = 1234h and B = 5678h. Figure 3 shows the "empty" stock, where the initial value of SP register = 100h and the stack starts at address of 58A10h and ends at address of 58B0Fh (100 bytes). Figure 4 shows the stack after two Push operations: Push A and Push B, and Figure 5 shows the stack after two Pushes and one Pop operations (Pop B).

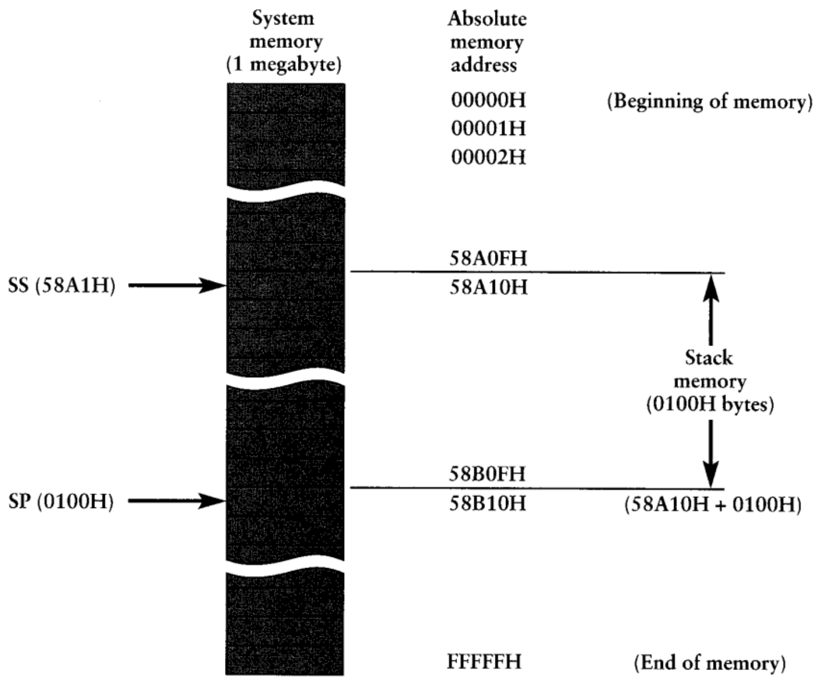


Figure 3: Empty stack

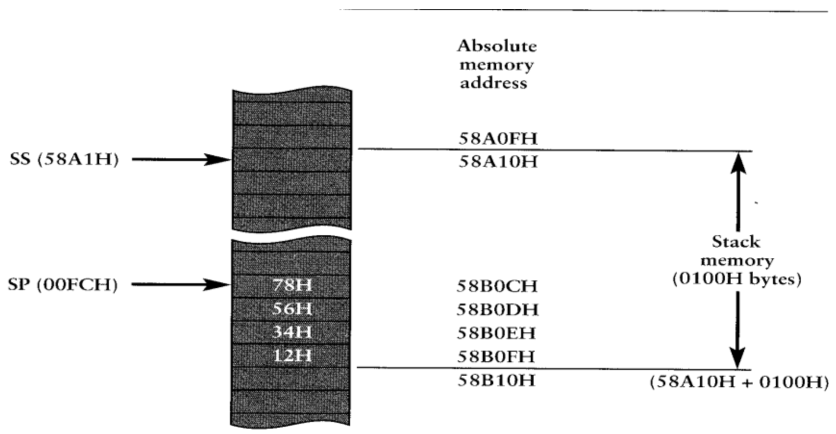


Figure 4: Stack after two Pushes

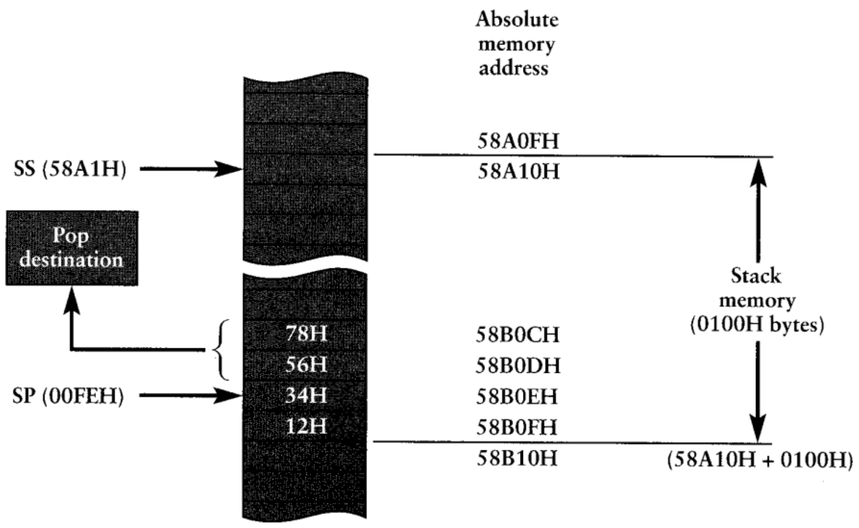


Figure 5: Stack after two Pushes and one Pop

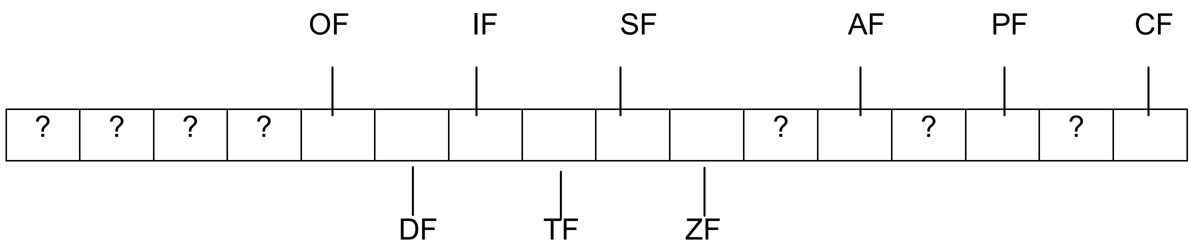
Internal Storage (Registers)

Registers: there are 14 registers, each 16 bits (bits 0-15) in length. 14 registers are divided into five groups:

1) Four segment registers
2) Four data registers
3) Two pointer and two index registers
4) The instruction pointer register
5) The flag register (status register)

Status register

In 8086/8088 the status register is called the flag register. The flag register contains information about the most recently executed instruction. There are 16 bits in register. But only bits 0, 2, 4 and 6-11 are used



1) CF (bit 0): the carry flag.
2) PF (bit 2): the parity flag.
3) AF (bit 4): the auxiliary flag.
4) ZF (bit 6): the zero flag.
5) SF (bit 7): the sign flag.
6) TF (bit 8): the trap flag.
7) IF (bit 9): the interrupt flag.
8) DF (bit 10): the direction flag.
9) OF (bit 11): the overflow flag.

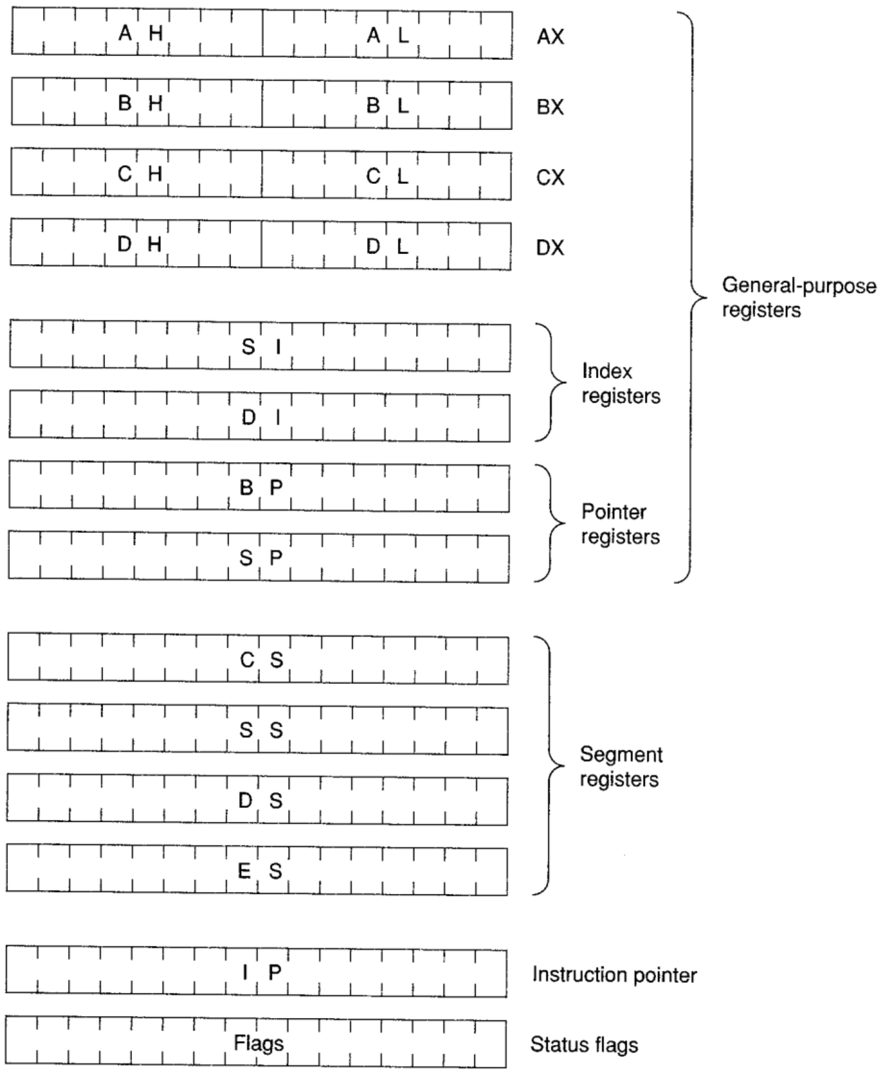


Figure 6: Register Map of 8086/8088

Segment Registers

The 8086/8088 has four separate "work area" for the parts of an assembly language program: data, instructio
data. Each area is called a segment and has a register to associate with called segment register. Each segn
65,536 bytes, and two segments my overlap.

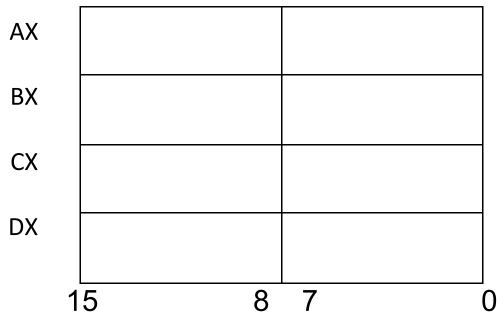
1) Data segment register (DS): indicates the beginning of the data segment.
2) Code segment register (CS): indicates the beginning of the code segment.
3) Stack segment register (SS): the stack is a part of memory. The stack is only word-addressable.
4) Extra data segment register (ES): points to the beginning of the extra data

Data Registers

There are four 16-bit data registers that are used to store data.

1) Accumulator: AX (used for arithmetic and logic instruction).

2) Base register: BX (stored data)
3) Count register: CX (is altered by the loop instruction).
4) Data register: DX (used for multiply and divide instructions). An “X” register can be subdivided into a high
and a low part (L, 0-7bits).



Pointer And Index Registers

The segment registers point to the beginning of the various memory segments. The point registers are used t
(displacements) into these segments.

- There are two pointer registers:
    1) Stack pointer (SP)
    2) Base pointer (BP)
- Stack pointer (SP): SP is used to hold the current stack location. SS contains the address of the begi
segment and SP contains the offset.
- A stack is a pile or list of items that can be accessed one at a time and from only one end, usually the i
from the top and added to the top. Thus, the stack is also called last-in first-out (LIFO) list. In8086/8
implemented by a segment of the memory, each item is a 16-bit word (2 bytes). There are two registe
stack: SS and SP.
- There are two basic stack operations in 8086/8088. a. PUSH (operand) it decreases the SP by 1 wo
stores a word on the stack e.g. current SP=204. AX=0010. Execution of PUSH AX will result: SP=20
0010 at memory location at address 202. b. POP (operand it removes the top word from the stack, and t
bytes) to the SP.
- The base pointer (BP): BP is also can be used to point to offset within stack segment. Primarily it is use
addressing modes.
- There are two index registers:
    1) Source index register (SI)

2) Destination index register (DI) there is no significance to the words source and destination, and
used interchangeably. Index registers are used in the same way subscripts that are used in hig
That enable us to access the elements in an array or table.

Instruction Pointer

The instruction pointer (IP) contains the offset address in the code segment of the next instruction to be execu
address of the next instruction = CS : IP.

UNIT 2 PROGRAMMING 8085 MICROPROCESSOR

8085 assembly language programming, addressing modes, 8085 instruction set, Instruction formats, Instruction Classification: data transfer, arithmetic operations, logical operations, branching operations, machine control —Stack and subroutines, Example Programs

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	Incrementregisteror 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

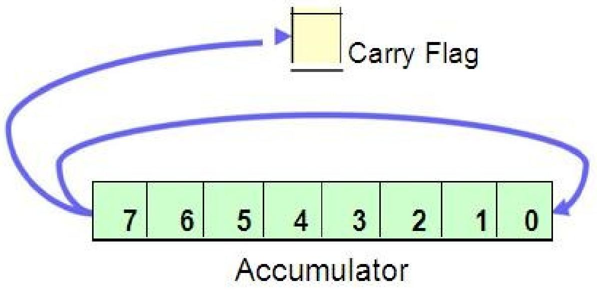


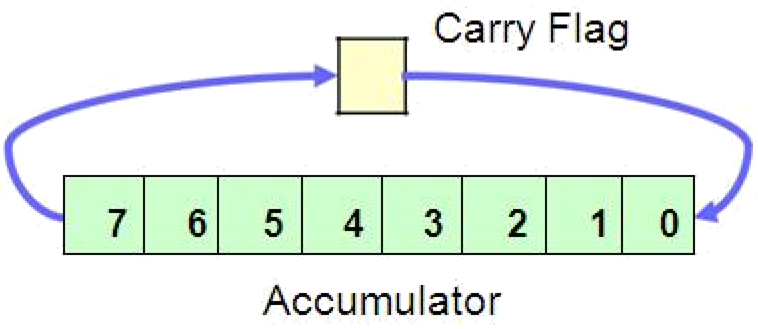
Fig 5. : Work flow of RLC

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



ig 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	Compareregisteror 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).

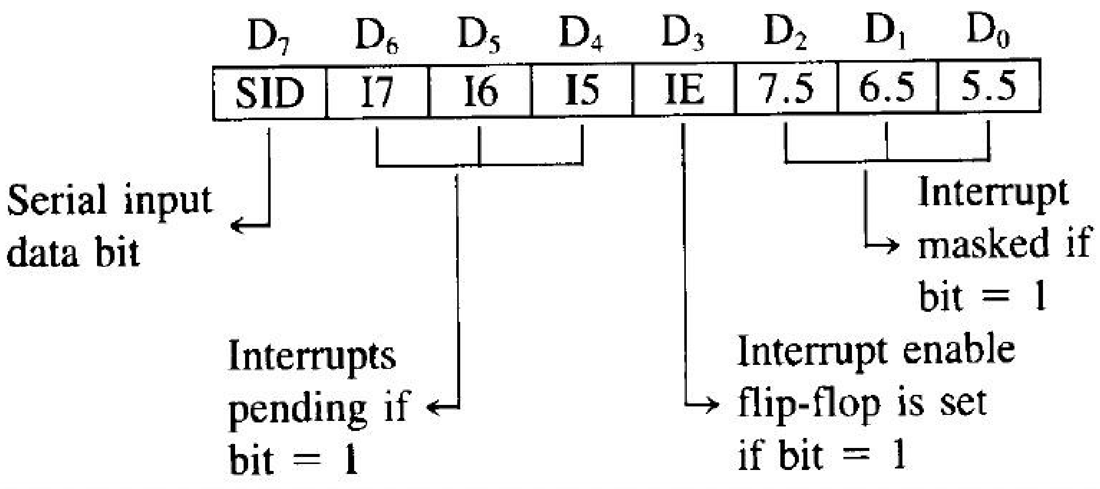
Example: EI

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

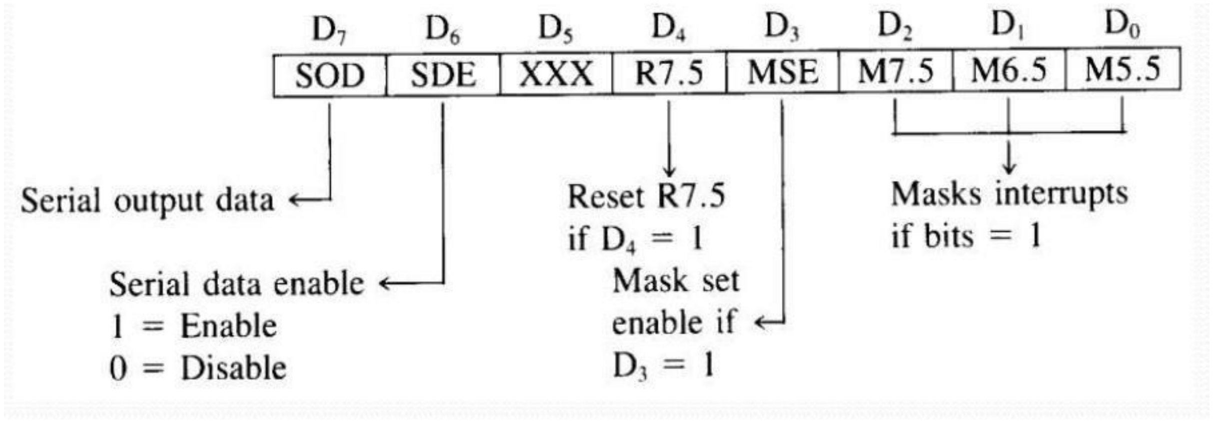

D_7
D_6
D_5
D_4
D_3
D_2
D_1
D_0
SID
I7
I6
I5
IE
7.5
6.5
5.5
Serial input data bit
Interrupts pending if bit = 1
Interrupt enable flip-flop is set if bit = 1
Interrupt masked if bit = 1

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



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.

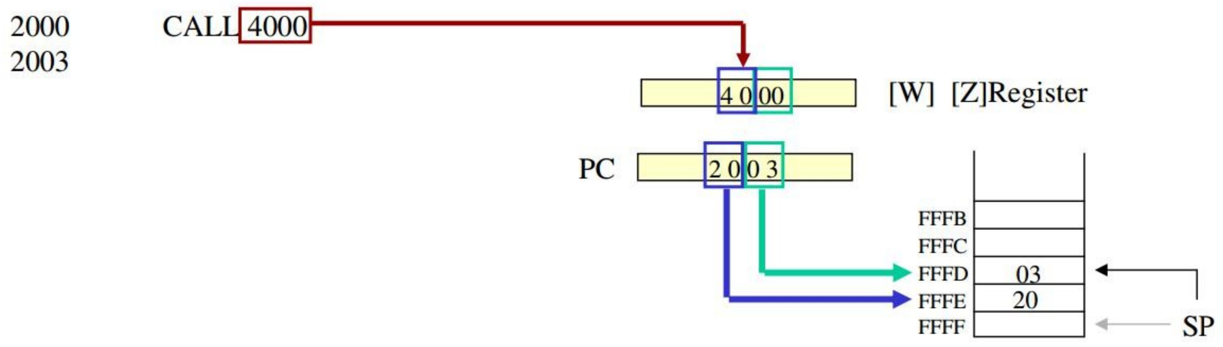


Fig 17.Work flow of CALL instruction

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.

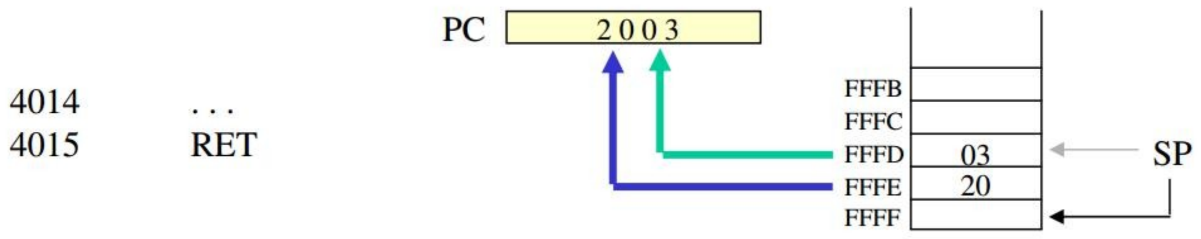


Fig 18.Work flow of RET instruction

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

8085 PROGRAMS

Programs:

ADDITION OF TWO 8 BIT NUMBERS

AIM:

To perform addition of two 8 bit numbers using 8085.

ALGORITHM:

1) Start the program by loading the first data into Accumulator.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Add the two register contents.
5) Check for carry.
6) Store the value of sum and carry in memory location.
7) Terminate the program.

PROGRAM:

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

OBSERVATION:

Input:	80 (4150)
	80 (4251)
Output:	00 (4152)
	01 (4153)

RESULT:

Thus the program to add two 8-bit numbers was executed.

SUBTRACTION OF TWO 8 BIT NUMBERS

AIM:

To perform the subtraction of two 8 bit numbers using 8085.

ALGORITHM:

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

PROGRAM:

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

OBSERVATION:

```
Input:  06 (4150)
        02 (4251)
Output: 04 (4152)
        01 (4153)
```

RESULT:

Thus the program to subtract two 8-bit numbers was executed.

<u>MULTIPLICATION OF TWO 8 BIT NUMBERS</u>

AIM:

To perform the multiplication of two 8 bit numbers using 8085.

ALGORITHM:

1) Start the program by loading HL register pair with address of memory location.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Add the two register contents.
5) Check for carry.
6) Increment the value of carry.
7) Check whether repeated addition is over and store the value of product and carry in memory location.
8) Terminate the program.

PROGRAM:

```
          MVI    D, 00     Initialize register D to 00
          MVI    A, 00     Initialize Accumulator content to 00
          LXI    H, 4150
          MOV    B, M      Get the first number in B - reg
          INX    H
          MOV    C, M      Get the second number in C- reg.
                           Add content of A - reg to register
 LOOP:    ADD    B         B.
          JNC    NEXT      Jump on no carry to NEXT.
          INR    D         Increment content of register D
 NEXT:    DCR    C         Decrement content of register C.
          JNZ    LOOP      Jump on no zero to address
          STA    4152      Store the result in Memory
          MOV    A, D
          STA    4153      Store the MSB of result in Memory
          HLT              Terminate the program.
```

OBSERVATION:

```
Input:      FF (4150)
            FF (4151)
Output:     01 (4152)
            FE (4153)
```

RESULT:Thus the program to multiply two 8-bit numbers was executed.

<u>DIVISION OF TWO 8 BIT NUMBERS</u>

AIM:

To perform the division of two 8 bit numbers using 8085.

ALGORITHM:

1) Start the program by loading HL register pair with address of memory location.
2) Move the data to a register(B register).
3) Get the second data and load into Accumulator.
4) Compare the two numbers to check for carry.
5) Subtract the two numbers.
6) Increment the value of carry .
7) Check whether repeated subtraction is over and store the value of product and carry in memory location.
8) Terminate the program.

PROGRAM:

```
         LXI    H, 4150
         MOV    B, M       Get the dividend in B – reg.
         MVI    C, 00      Clear C – reg for qoutient
         INX    H
         MOV    A, M       Get the divisor in A – reg.
                           Compare A - reg with register
NEXT:    CMP    B          B.
         JC     LOOP       Jump on carry to LOOP
         SUB    B          Subtract A – reg from B- reg.
                           Increment content of register
         INR    C          C.
         JMP    NEXT       Jump to NEXT
                           Store the remainder in
LOOP:    STA    4152       Memory
         MOV    A, C
         STA    4153       Store the quotient in memory
         HLT               Terminate the program.
```

OBSERVATION:

Input: FF (4150)
FF (4251)

```
Output:     01 (4152) ---- Remainder
            FE (4153) ---- Quotient
```

RESULT:

Thus the program to divide two 8-bit numbers was executed.

LARGEST NUMBER IN AN ARRAY OF DATA

AIM:

To find the largest number in an array of data using 8085 instruction set.

ALGORITHM:

1) Load the address of the first element of the array in HL pair
2) Move the count to B – reg.
3) Increment the pointer
4) Get the first data in A – reg.
5) Decrement the count.
6) Increment the pointer
7) Compare the content of memory addressed by HL pair with that of A - reg.
8) If Carry = 0, go to step 10 or if Carry = 1 go to step 9
9) Move the content of memory addressed by HL to A – reg.
10) Decrement the count
11) Check for Zero of the count. If ZF = 0, go to step 6, or if ZF = 1 go to next step.
12) Store the largest data in memory.
13) Terminate the program.

PROGRAM:

```
        LXI     H,4200      Set pointer for array
        MOV     B,M         Load the Count
        INX     H
        MOV     A,M         Set 1st element as largest data
        DCR     B           Decrement the count
LOOP:   INX     H
        CMP     M           If A- reg > M go to AHEAD
        JNC     AHEAD
        MOV     A,M        Set the new value as largest
AHEAD:  DCR     B
                           Repeat comparisons till count =
        JNZ     LOOP       0
        STA     4300       Store the largest value at 4300
        HLT
```

OBSERVATION:

```
        Input:      05 (4200) ----- Array Size
                    0A (4201)
                    F1 (4202)

                    1F (4203)
                    26 (4204) FE
                    (4205)

                    FE
        Output:     (4300)
```

RESULT:

Thus the program to find the largest number in an array of data was executed

SMALLEST NUMBER IN AN ARRAY OF DATA

AIM:

To find the smallest number in an array of data using 8085 instruction set.

ALGORITHM:

1) Load the address of the first element of the array in HL pair
2) Move the count to B – reg.
3) Increment the pointer
4) Get the first data in A – reg.
5) Decrement the count.
6) Increment the pointer
7) Compare the content of memory addressed by HL pair with that of A - reg.
8) If carry = 1, go to step 10 or if Carry = 0 go to step 9
9) Move the content of memory addressed by HL to A – reg.
10) Decrement the count
11) Check for Zero of the count. If ZF = 0, go to step 6, or if ZF = 1 go to next step.

12) Store the smallest data in memory.
13) Terminate the program.

PROGRAM:

```
                                 Set pointer for
        LXI         H,4200       array
        MOV         B,M          Load the Count
        INX         H

        MOV         A,M          Set 1st element as largest data
        DCR         B            Decrement the count
LOOP:   INX         H
        CMP         M            If A- reg < M go to AHEAD
        JC          AHEAD
        MOV         A,M          Set the new value as smallest
AHEAD:  DCR         B
                                 Repeat comparisons till count =
        JNZ         LOOP         0
        STA         4300         Store the largest value at 4300
        HLT
```

OBSERVATION:

```
        Input:      05 (4200) ----- Array Size
                    0A (4201)
                    F1 (4202)
                    1F (4203)
                    26 (4204)
                    FE (4205)

        Output:     0A (4300)
```

RESULT:

Thus the program to find the smallest number in an array of data was executed

<u>ARRANGE AN ARRAY OF DATA IN ASCENDING ORDER</u>

AIM:

To write a program to arrange an array of data in ascending order

ALGORITHM:

1. Initialize HL pair as memory pointer
2. Get the count at 4200 into C – register
3. Copy it in D – register (for bubble sort (N-1) times required)
4. Get the first value in A – register
5. Compare it with the value at next location.
6. If they are out of order, exchange the contents of A –register and Memory
7. Decrement D –register content by 1
8. Repeat steps 5 and 7 till the value in D- register become zero
9. Decrement C –register content by 1
10. Repeat steps 3 to 9 till the value in C – register becomes zero

PROGRAM:

```
                LXI         H,4200
                MOV         C,M
                DCR         C
REPEAT:         MOV         D,C
                LXI         H,4201
LOOP:           MOV         A,M
                INX         H
                CMP         M
                JC          SKIP
                MOV         B,M
                MOV         M,A
                DCX         H
                MOV         M,B
                INX         H
SKIP:           DCR         D
                JNZ         LOOP
                DCR         C
                JNZ         REPEAT
                HLT
```

OBSERVATION:

Input:	4200	05 (Array Size)
	4201	05
	4202	04
	4203	03
	4204	02
	4205	01

Output:	4200	05(Array Size)
	4201	01
	4202	02
	4203	03
	4204	04
	4205	05

RESULT:

Thus the given array of data was arranged in ascending order.

ARRANGE AN ARRAY OF DATA IN DESCENDING ORDER

AIM:

To write a program to arrange an array of data in descending order

ALGORITHM:

1. Initialize HL pair as memory pointer
2. Get the count at 4200 into C – register
3. Copy it in D – register (for bubble sort (N-1) times required)
4. Get the first value in A – register
5. Compare it with the value at next location.
6. If they are out of order, exchange the contents of A –register and Memory
7. Decrement D –register content by 1
8. Repeat steps 5 and 7 till the value in D- register become zero
9. Decrement C –register content by 1
10. Repeat steps 3 to 9 till the value in C – register becomes zero

PROGRAM:

```
        LXI     H,4200
```

```
            MOV       C,M
            DCR       C
REPEAT:     MOV       D,C
                      H,420
            LXI       1
LOOP:       MOV       A,M
            INX       H
            CMP       M
            JNC       SKIP
            MOV       B,M
            MOV       M,A
            DCX       H
            MOV       M,B
            INX       H
SKIP:       DCR       D
            JNZ       LOOP
                DCR       C
                JNZ       REPEAT
                HLT
```

OBSERVATION:

Input:	4200	05 (Array Size)
	4201	01
	4202	02
	4203	03
	4204	04
	4205	05

Output:	4200	05(Array Size)
	4201	05
	4202	04
	4203	03
	4204	02
	4205	01

RESULT:

Thus the given array of data was arranged in descending order.

UNIT 3 PERIPHERALS AND INTERFACING 9 Hrs.

Introduction, memory and I/O interfacing, data transfer schemes, Interface ICs'- USART (8251), programmable peripheral interface (8255), programmable interrupt controller (8259), programmable counter/interval timer (8254), Analog to Digital Converter (ADC), and Digital to Analog Converter (DAC).

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 some what complex. It is an important general purpose I/O device that can be used with almost any microprocessor.

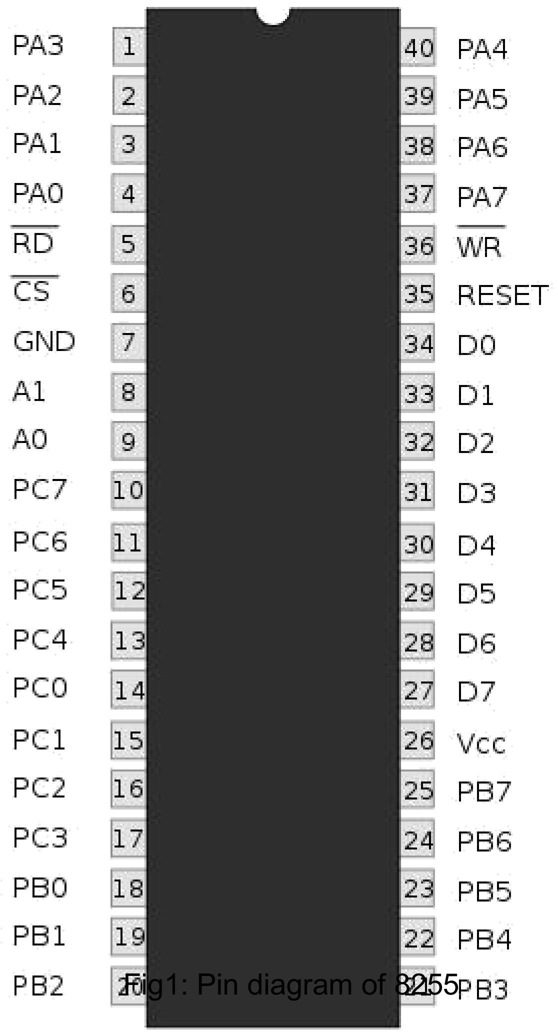
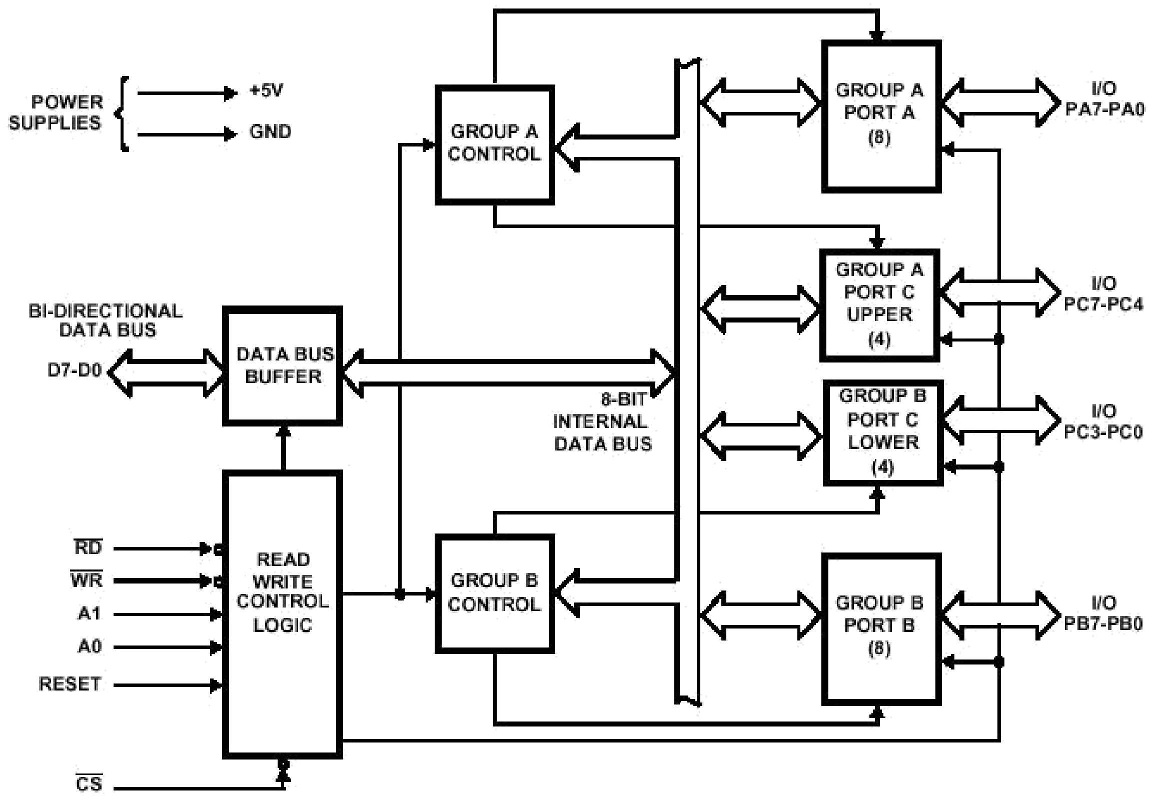


Fig1: Pin diagram of 8255

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 3



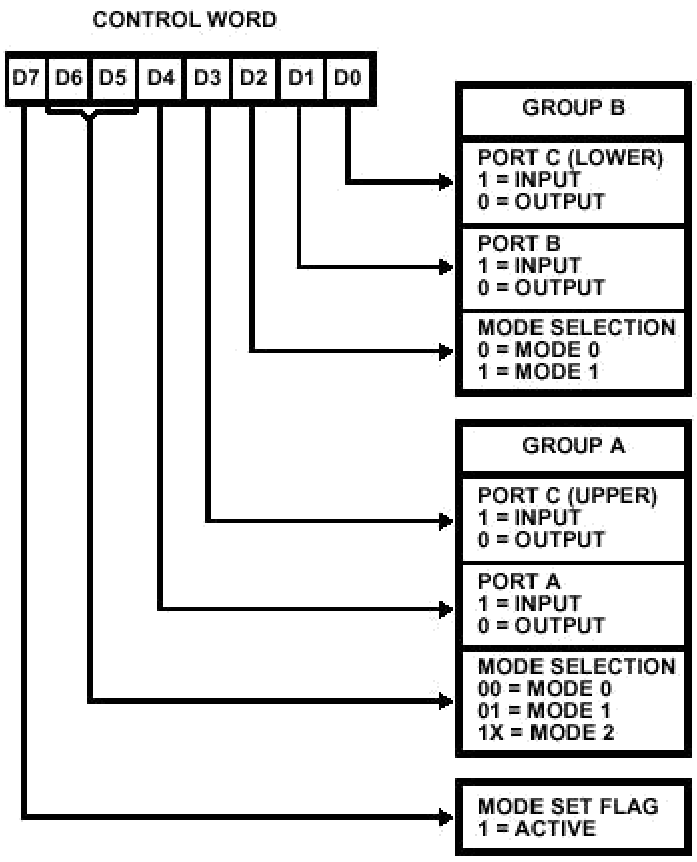


Fig 3. Control word Register format

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.

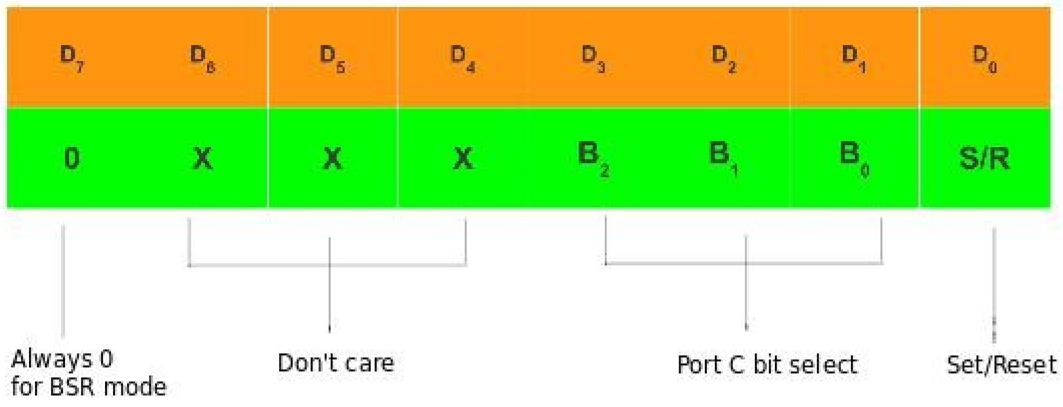


Fig 4: 8255 Control register format for BSR mode

- 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

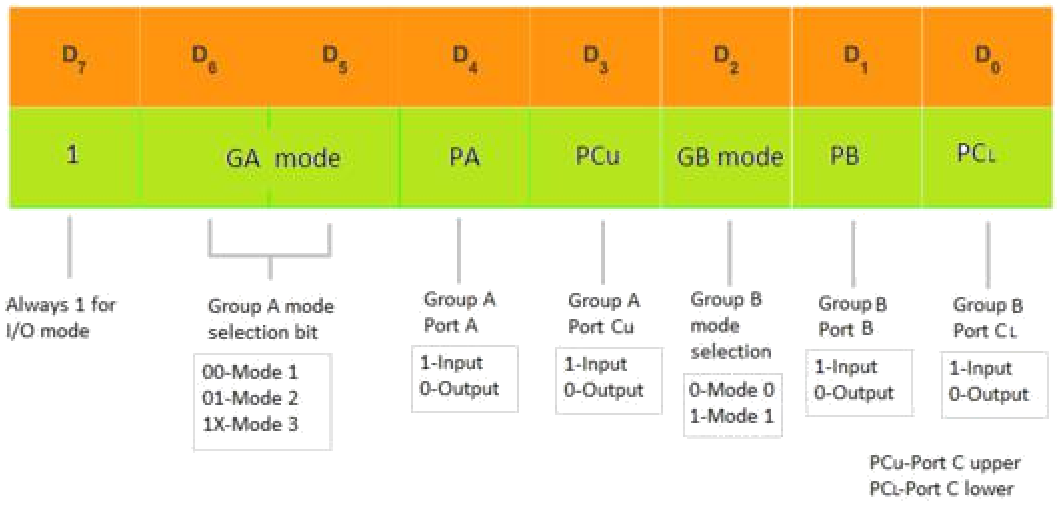


Figure 5: 8255 Control word for I/O mode

- **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

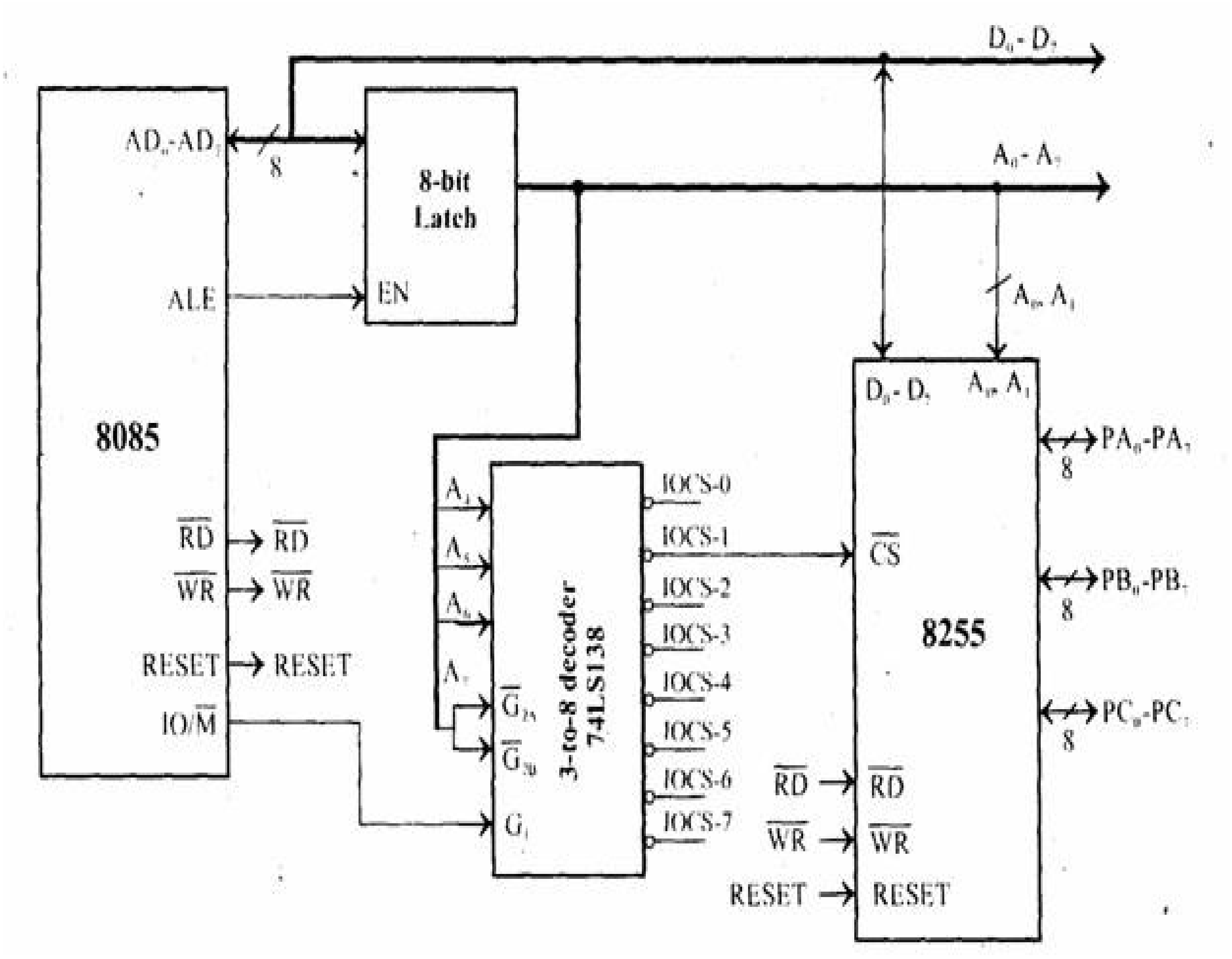


Fig 6. Interfacing 8255 with 8085 processor

- 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.

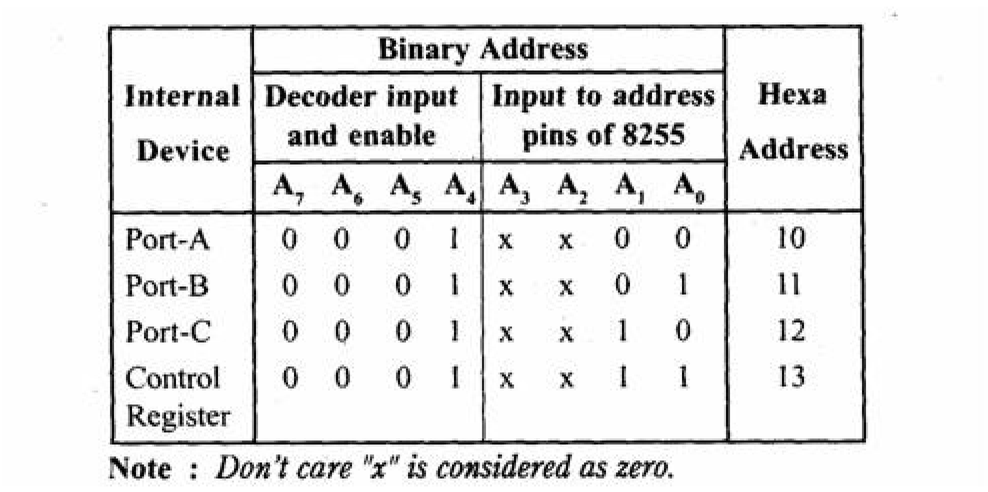
- Using a 3-to-8 decoder generates the chip select signals for I/O mapped devices.

- 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.

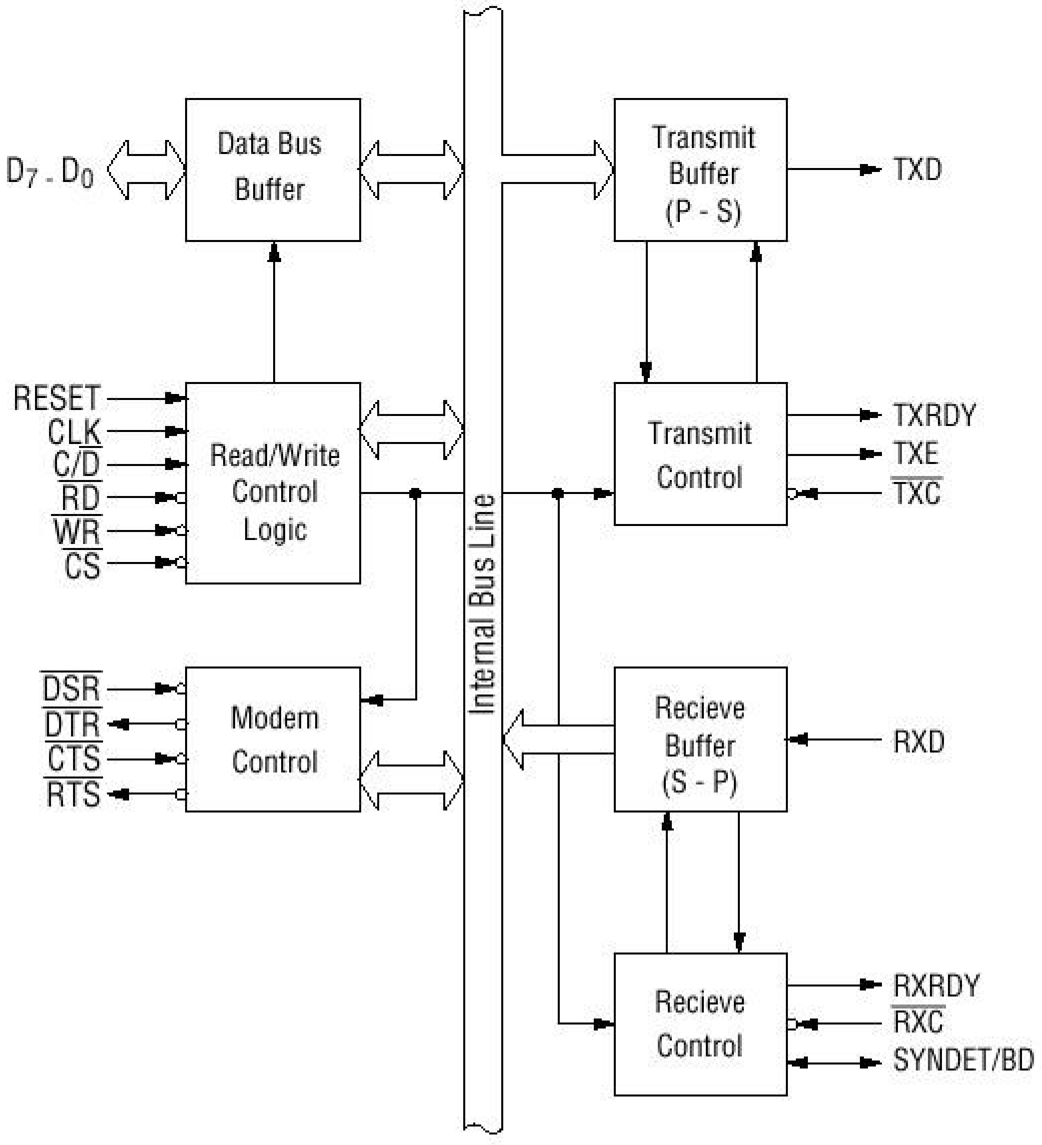


Figure 10 : Architecture of 8251

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.

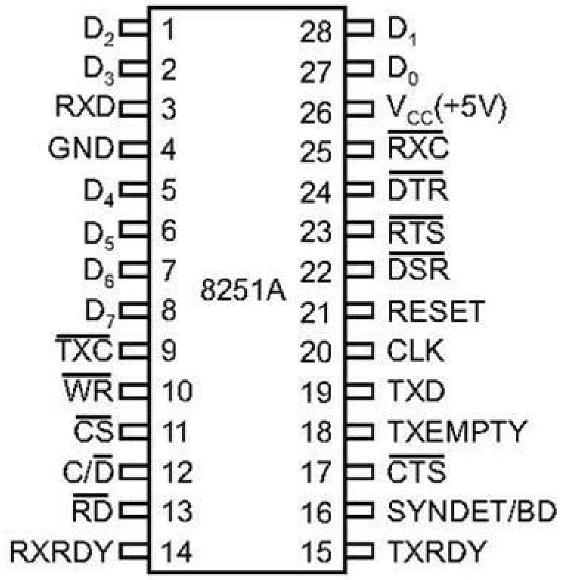


Fig 11 : Pin Configuration of 8251

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.

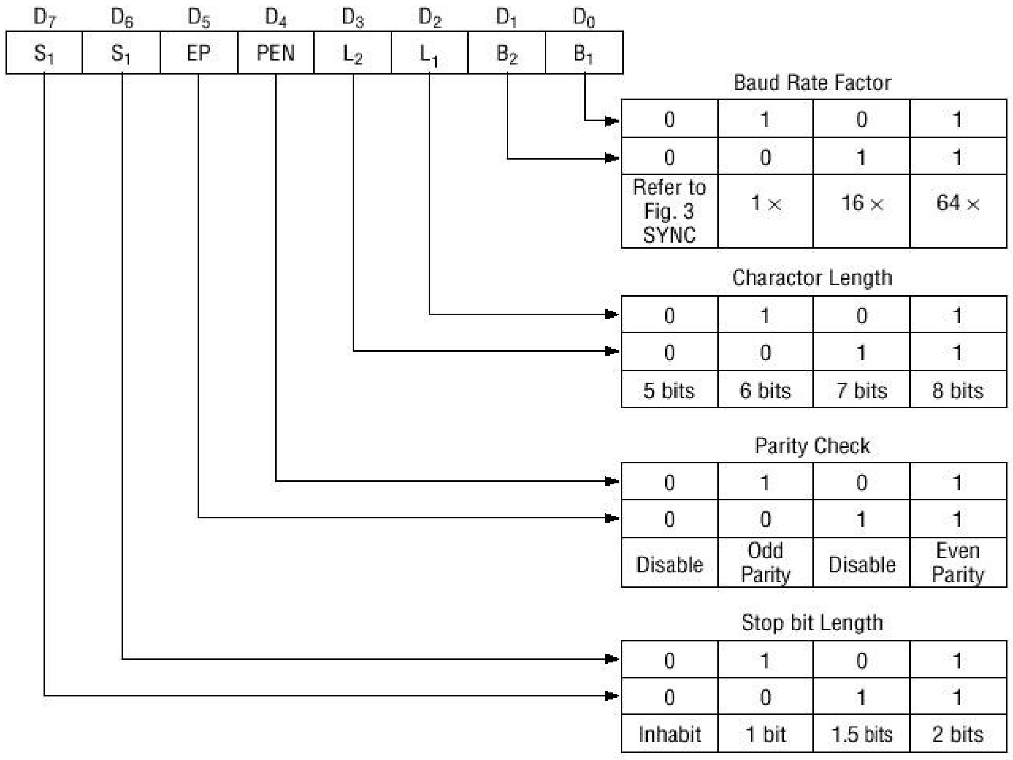


Fig. 2 Bit Configuration of Mode Instruction (Asynchronous)

Fig 12: Bit configuration of mode instruction(asynchronous)

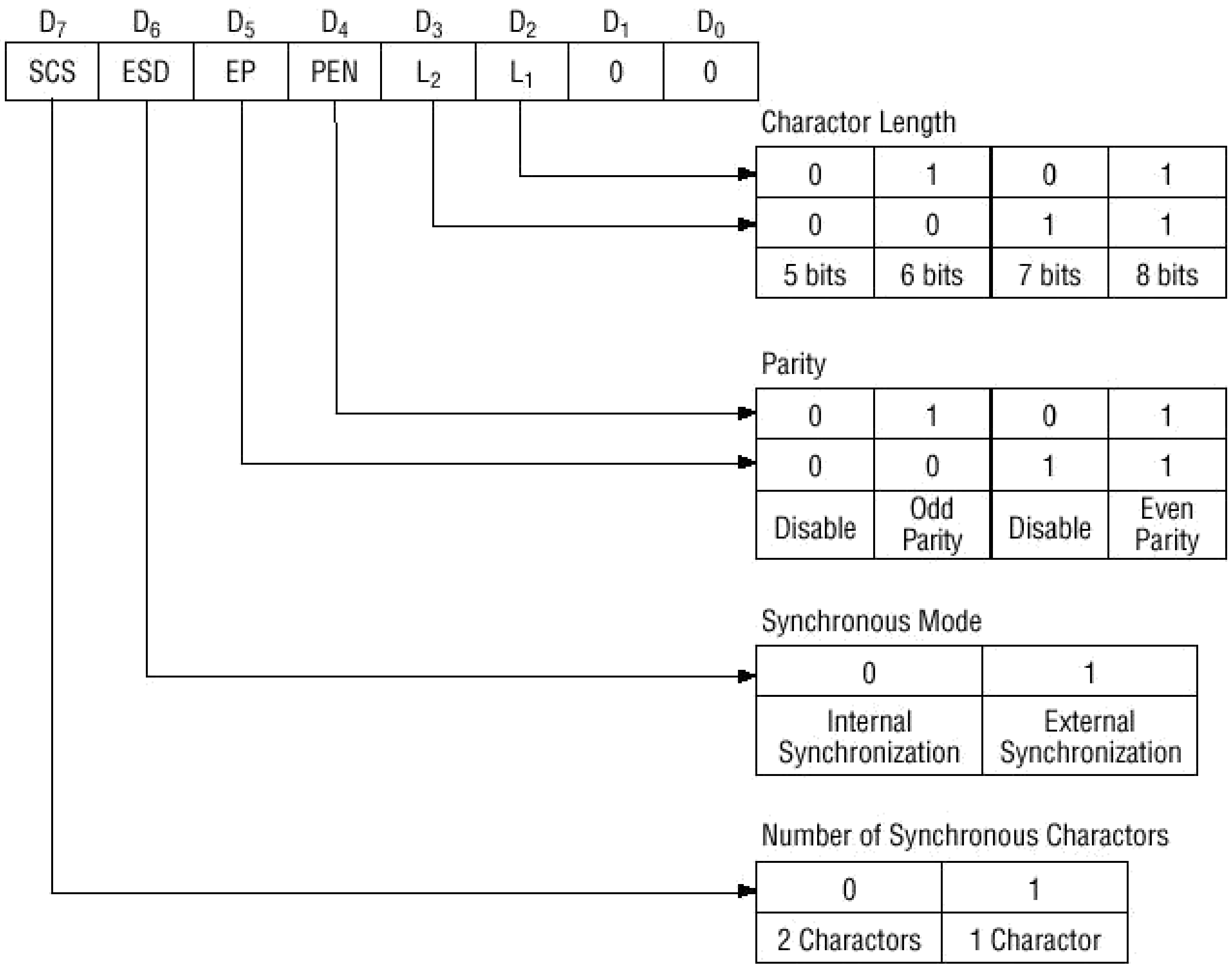


Fig. 3 Bit Configuration of Mode Instruction (Synchronous)

Fig 13: 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)

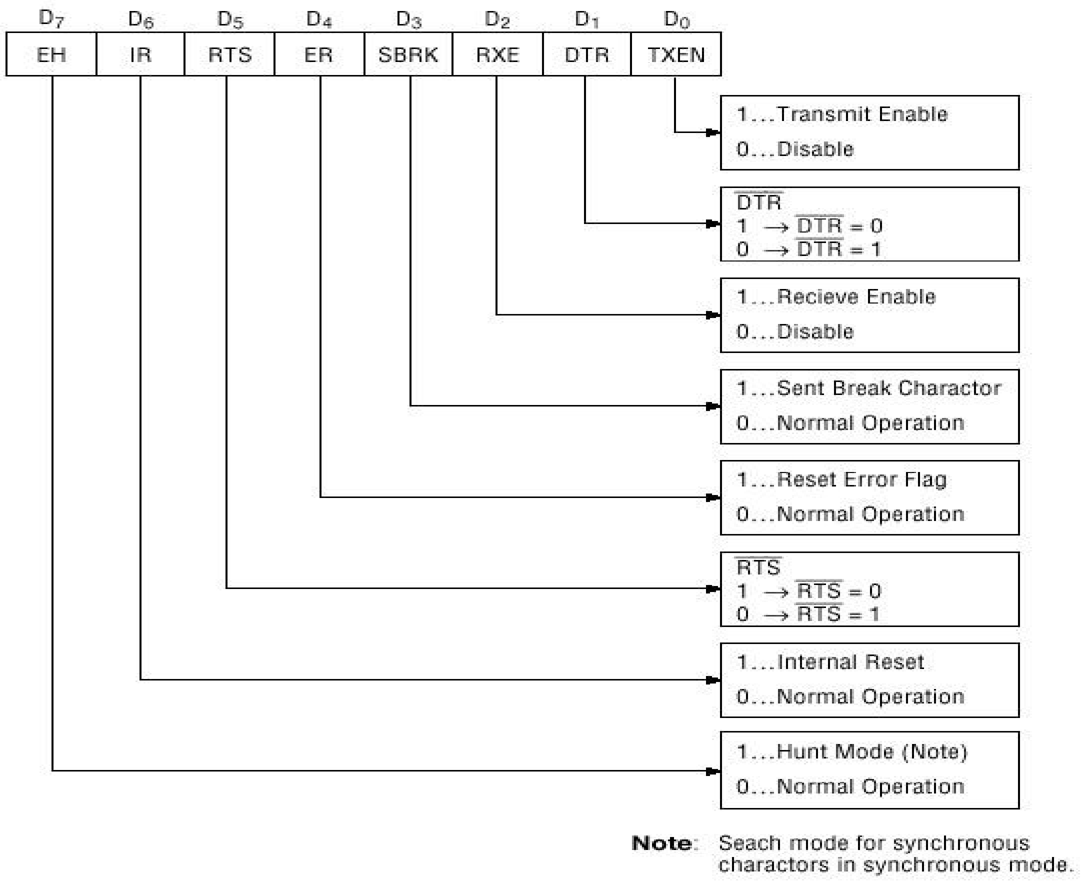


Fig. 4 Bit Configuration of Command

Fig 14: Bit configuration of command

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.

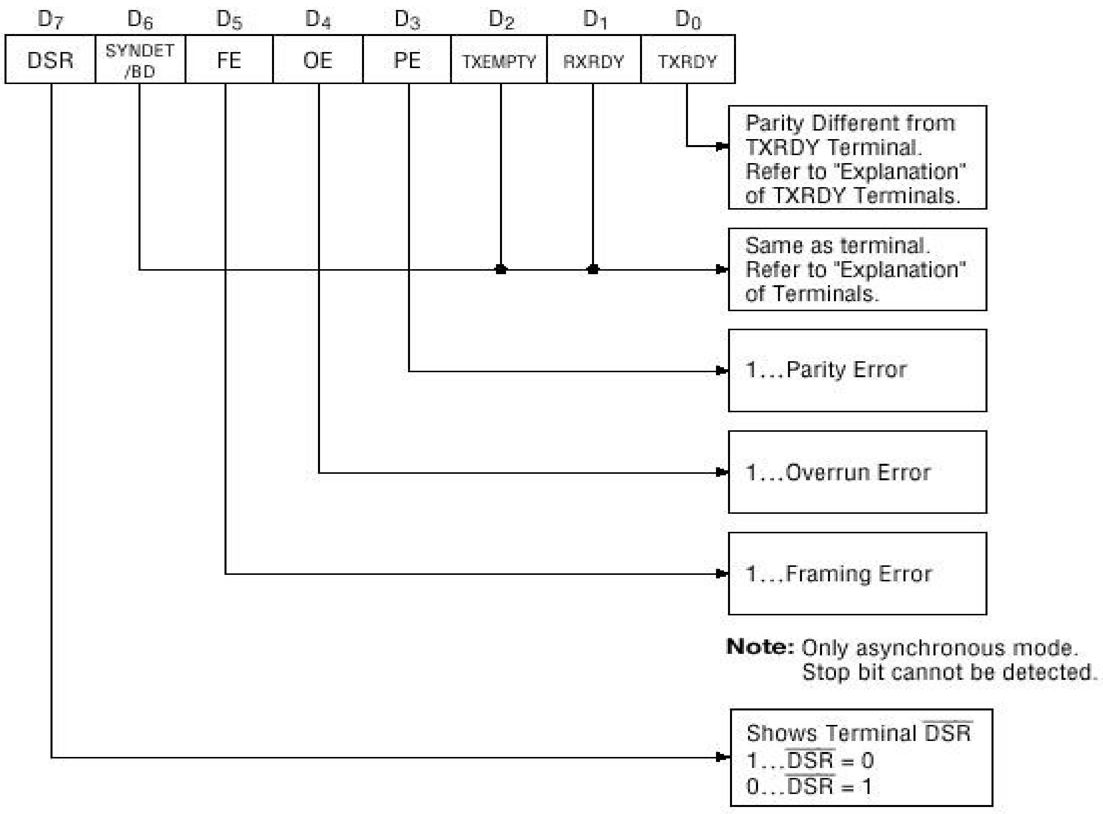

D7
D6
D5
D4
D3
D2
D1
D0
DSR
SYNDET /BD
FE
OE
PE
TXEMPTY
RXRDY
TXRDY
Parity Different from TXRDY Terminal. Refer to "Explanation" of TXRDY Terminals.
Same as terminal. Refer to "Explanation" of Terminals.
1...Parity Error
1...Overrun Error
1...Framing Error
Note: Only asynchronous mode. Stop bit cannot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

Fig. 5 Bit Configuration of Status Word

Fig 15: 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.

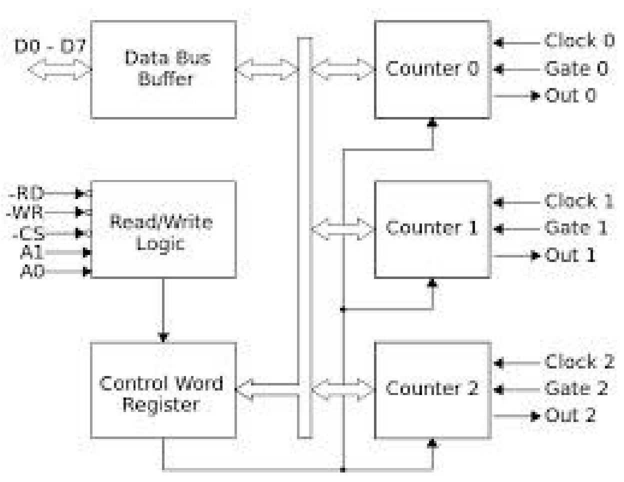


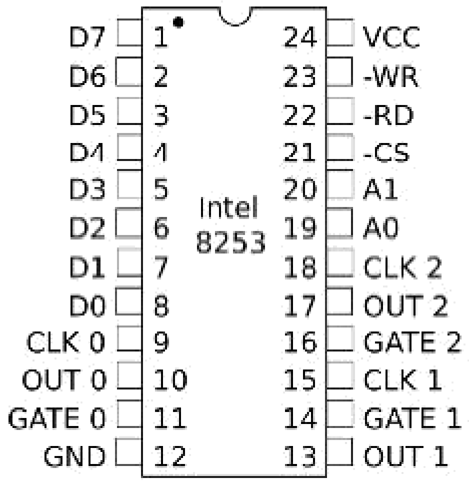
Fig : 16 Block Diagram of 8253 programmable interval timer

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.



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.

$\overline{CS}$:

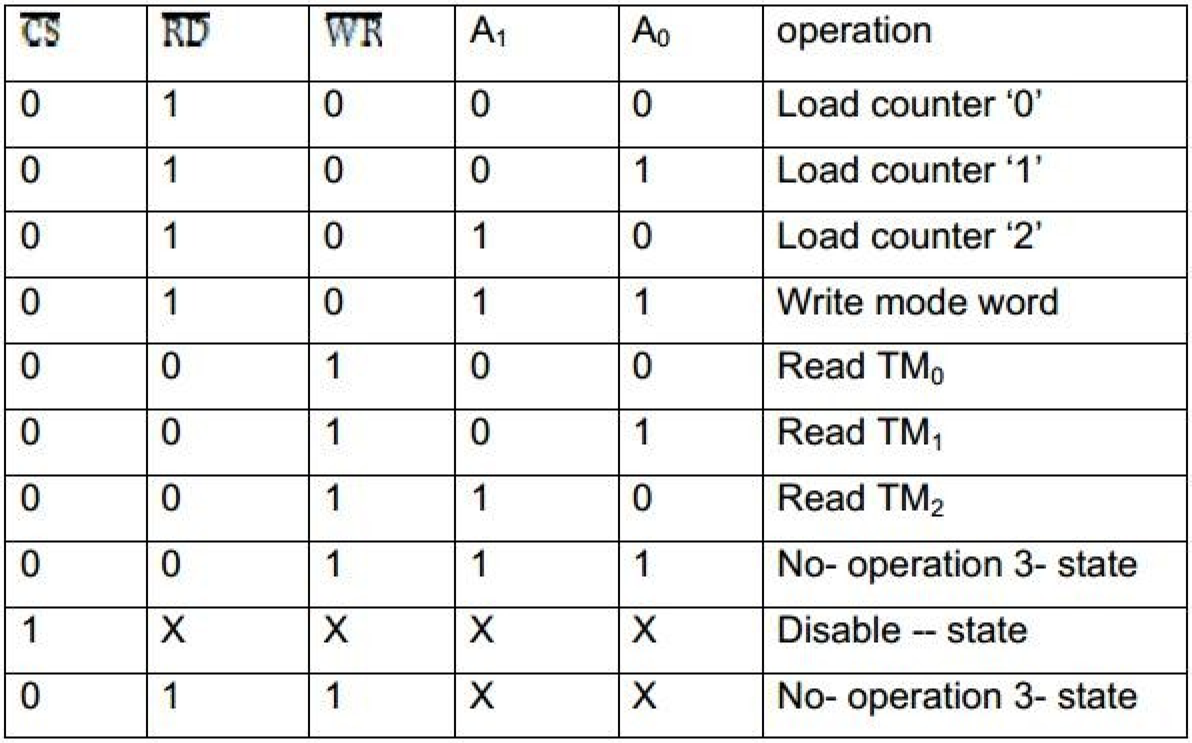
The chip select input is used to enable the communicate between 8253 and the microprocessor by means of data bus. A low an $\overline{CS}$ enables the data bus buffers, while a high disables the buffer. The $\overline{CS}$ input does not have any affect on the operation of three times once they have 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 ($\overline{RD}$) and write $\overline{WR}$ pins central the direction of data transfer on the 8-bit bus. When the input $\overline{RD}$ 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

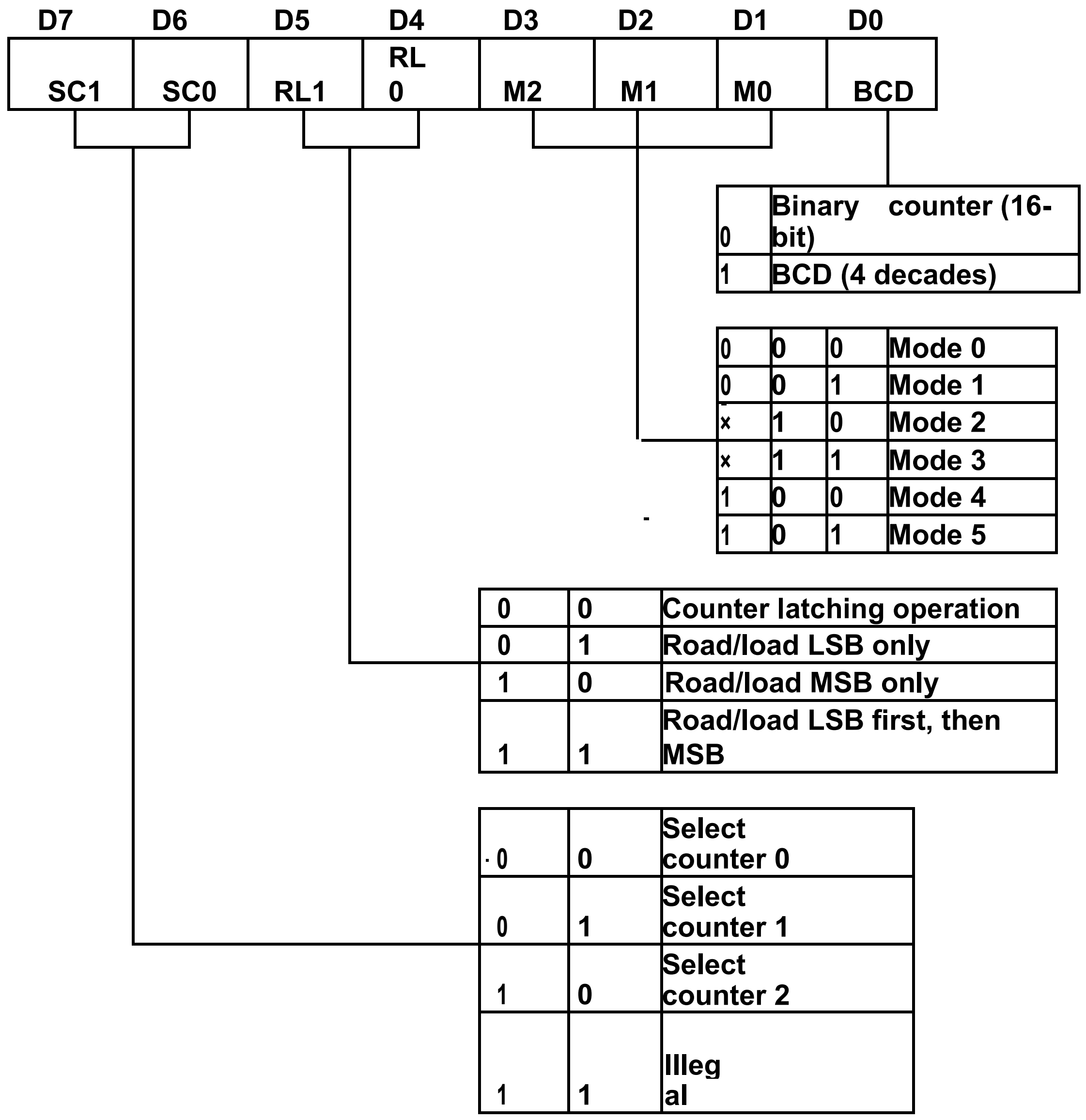
effect of the gate signal depends on which of the six modes of

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



D7	D6	D5	D4	D3	D2	D1	D0
SC1	**SC0**	**RL1**	**RL 0**	**M2**	**M1**	**M0**	**BCD**

0	**Binary counter (16-bit)**
1	**BCD (4 decades)**

0	0	0	**Mode 0**
0	0	1	**Mode 1**
×	1	0	**Mode 2**
×	1	1	**Mode 3**
1	0	0	**Mode 4**
1	0	1	**Mode 5**

0	0	**Counter latching operation**
0	1	**Road/load LSB only**
1	0	**Road/load MSB only**
1	1	**Road/load LSB first, then MSB**

0	0	**Select counter 0**
0	1	**Select counter 1**
1	0	**Select counter 2**
1	1	**Illeg al**

Control Register

MODES OF OPERATION

Mode 0 Interrupt on terminal count

Mode 1 Programmable one shot

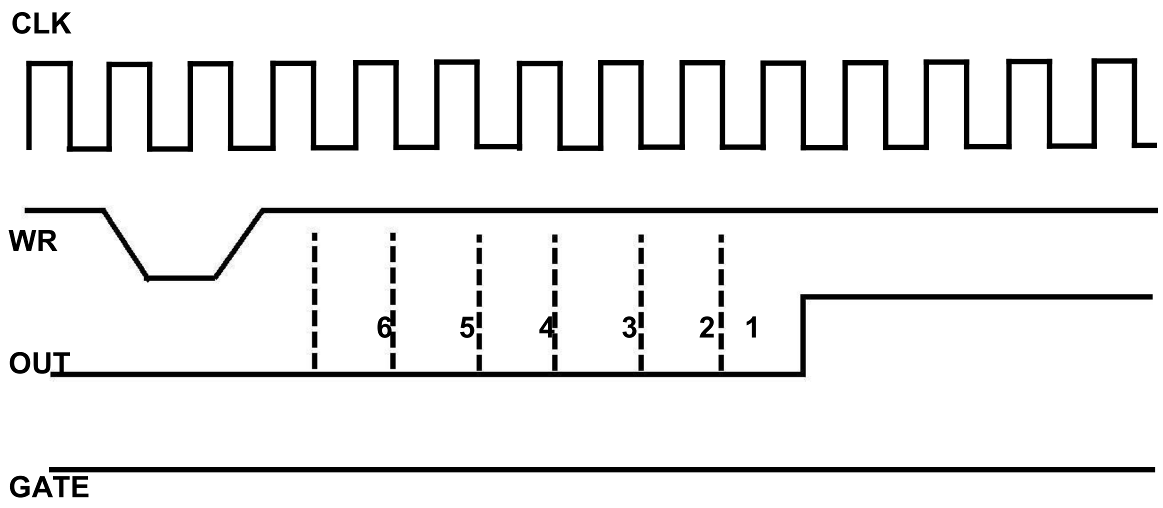
Mode 2 Rate Generator

Mode 3 Square wave rate Generator

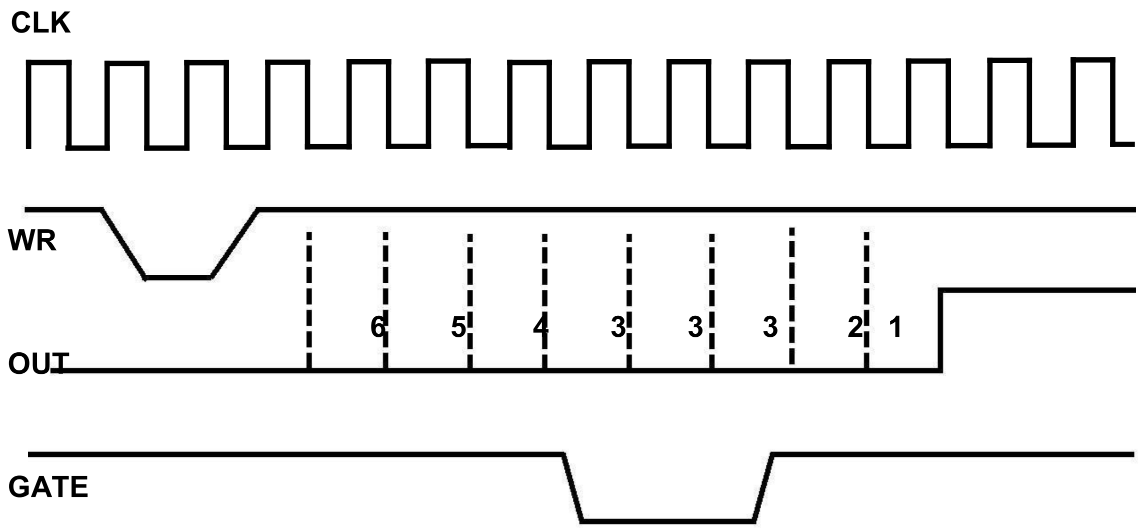
Mode 4 Software triggered strobe

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 .



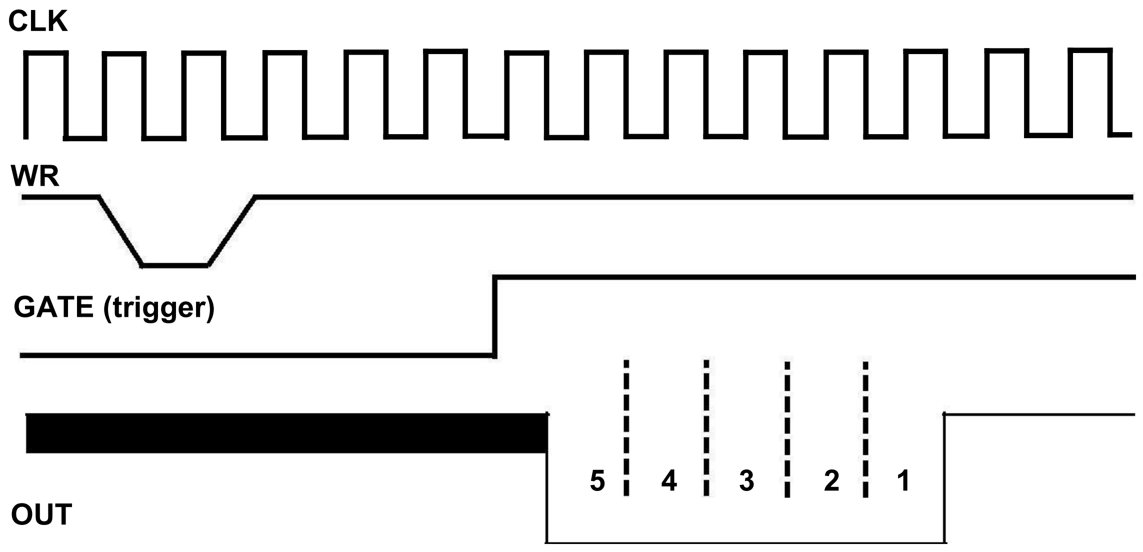
Mode 0 count when Gate is high (enabled)



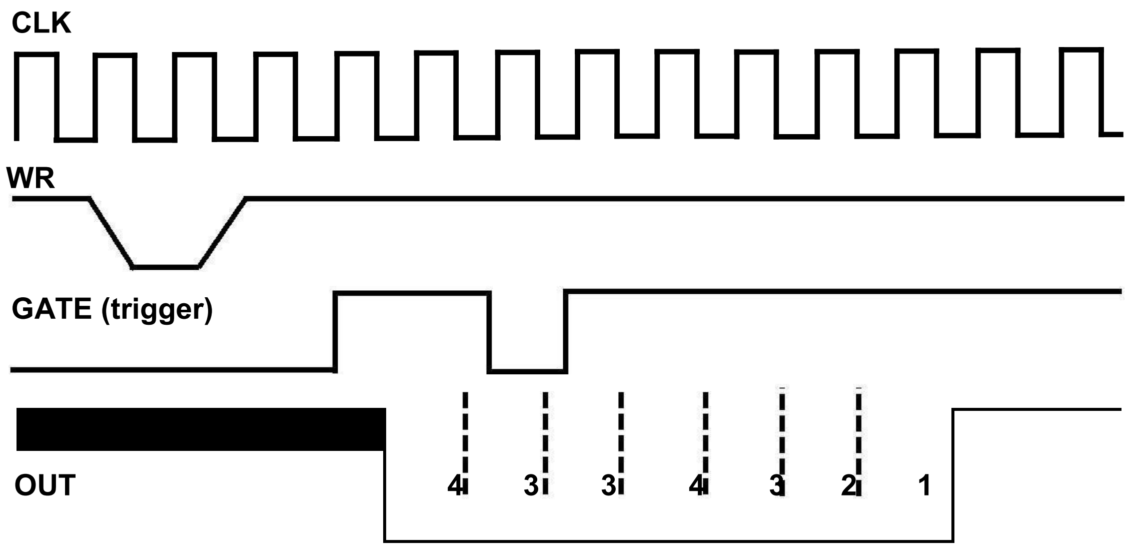
Mode 0 count when Gate is low temporarily (disabled)

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.



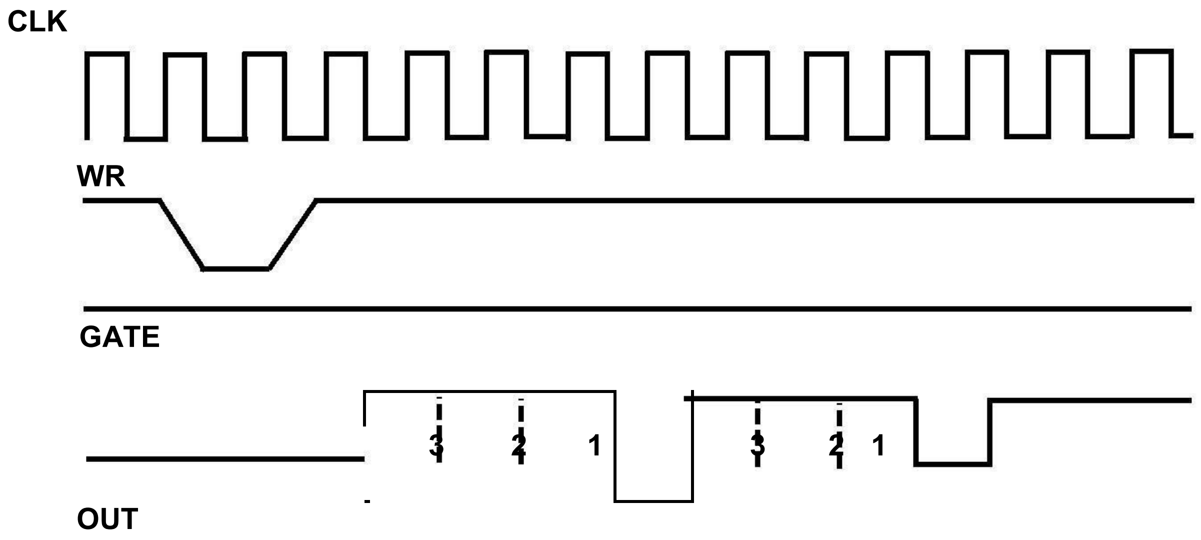
Mode 1 The Gate goes high. The output goes low for the period depending on the count



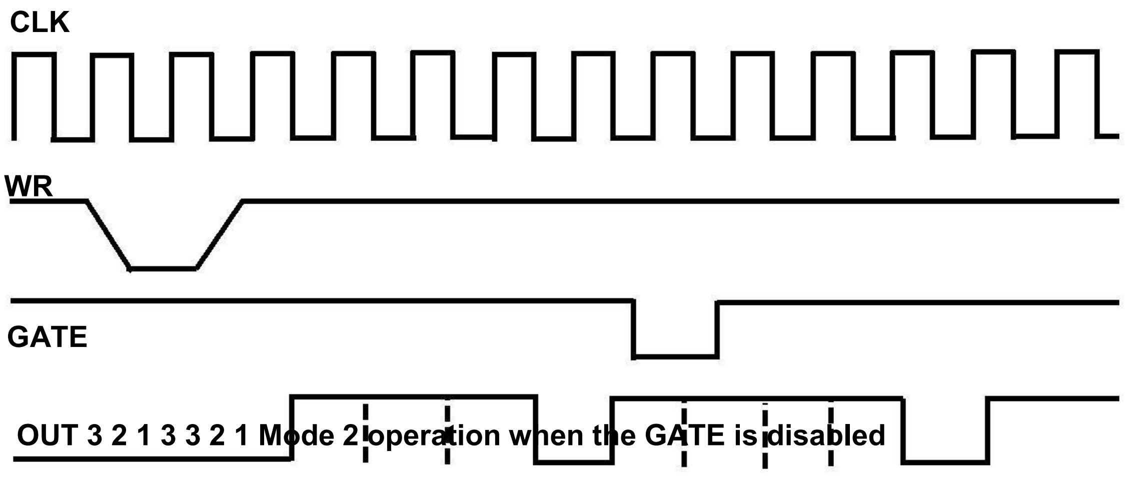
Mode 1 The Gate pulse is disabled momentarily causing the counter to stop.

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.



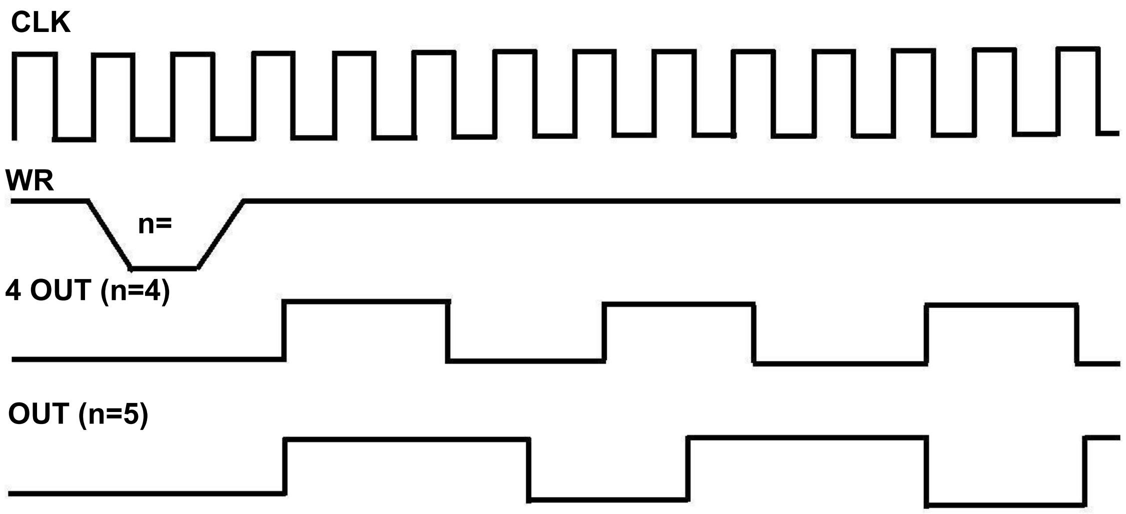
Mode 2 Operation when the GATE is kept high



Mode 2 operation when the GATE is disabled momentarily.

Mode 3 Programmable Square Wave Rate Generator

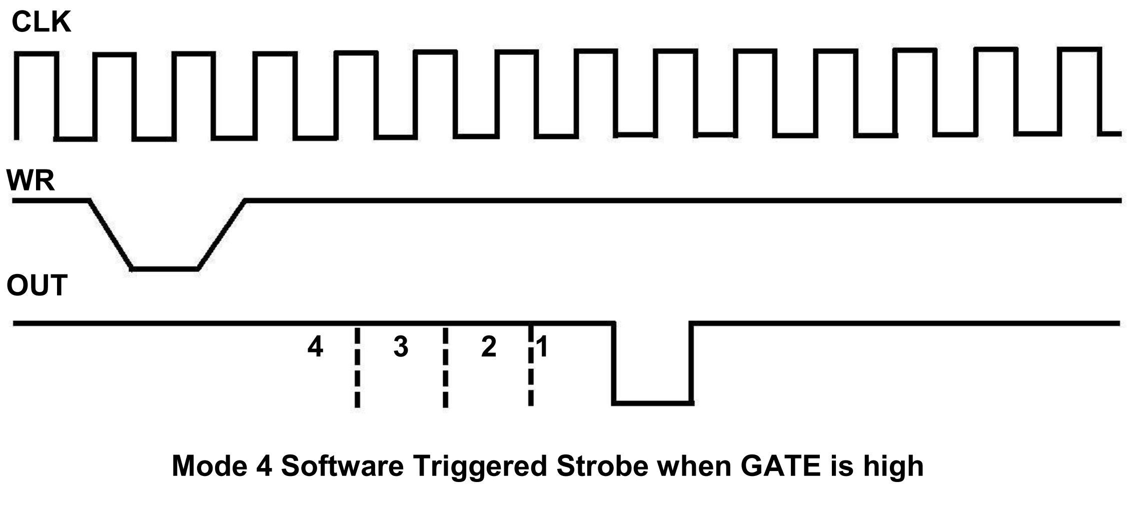
It is similar to Mode 2 but the output high and low period is symmetrical. The output goes high after the count is loaded and it remains high for period which equals the count down period of the counter register. The output subsequently goes low for an equal period and hence generates a symmetrical square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



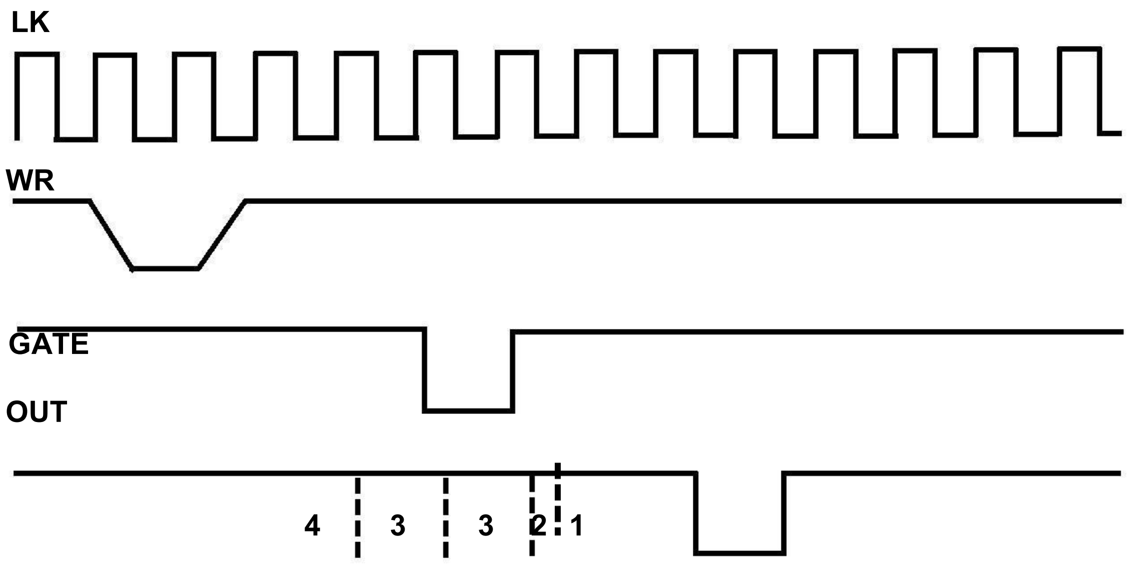
Mode3 Operation: Square Wave generator

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.



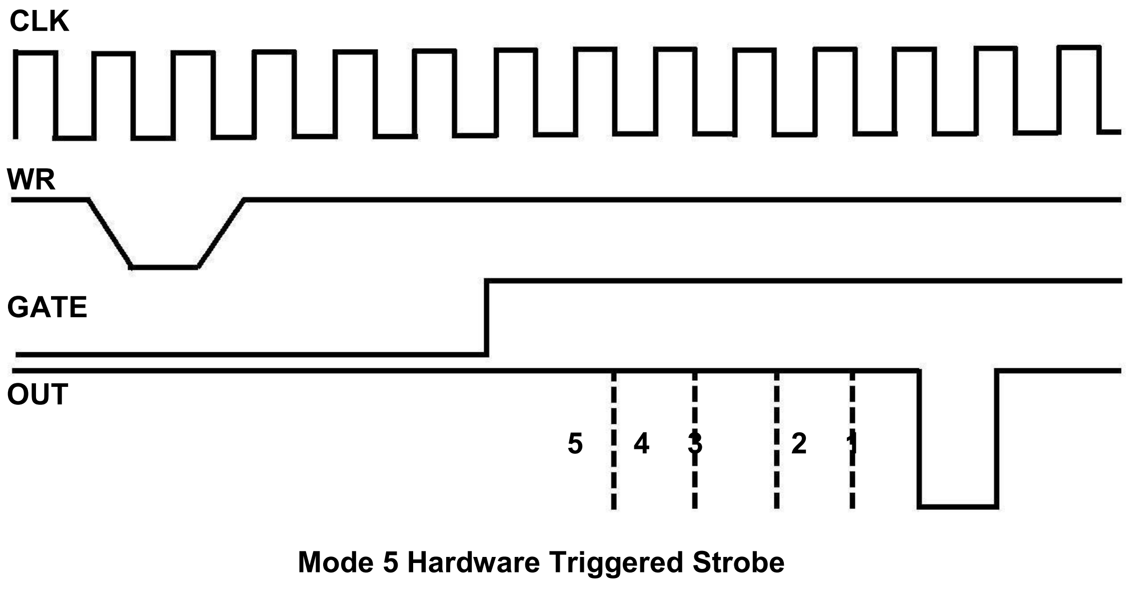
Mode 4 Software Triggered Strobe when GATE is high



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.



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
    - oStandard Temperature Range
    - oExtended 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.

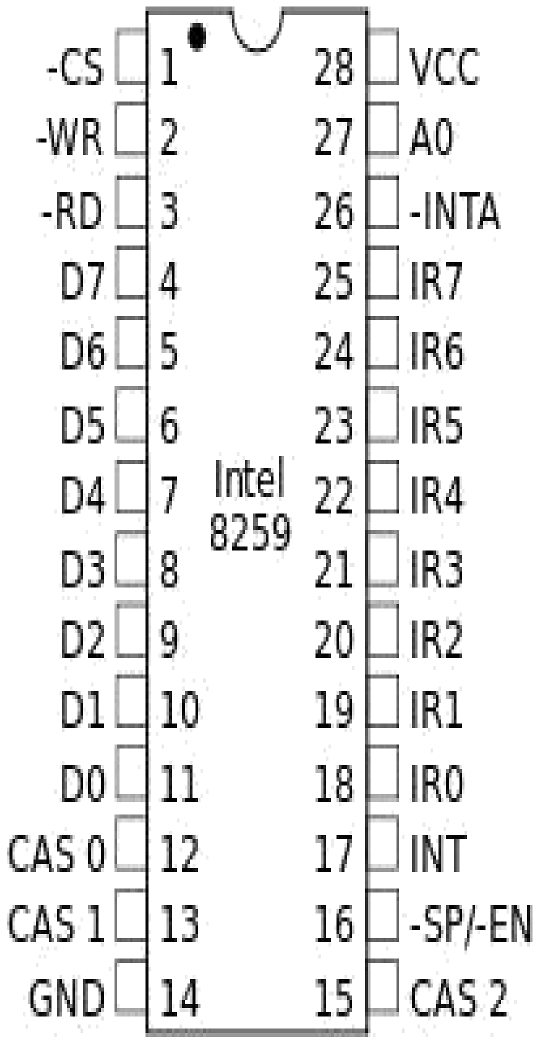
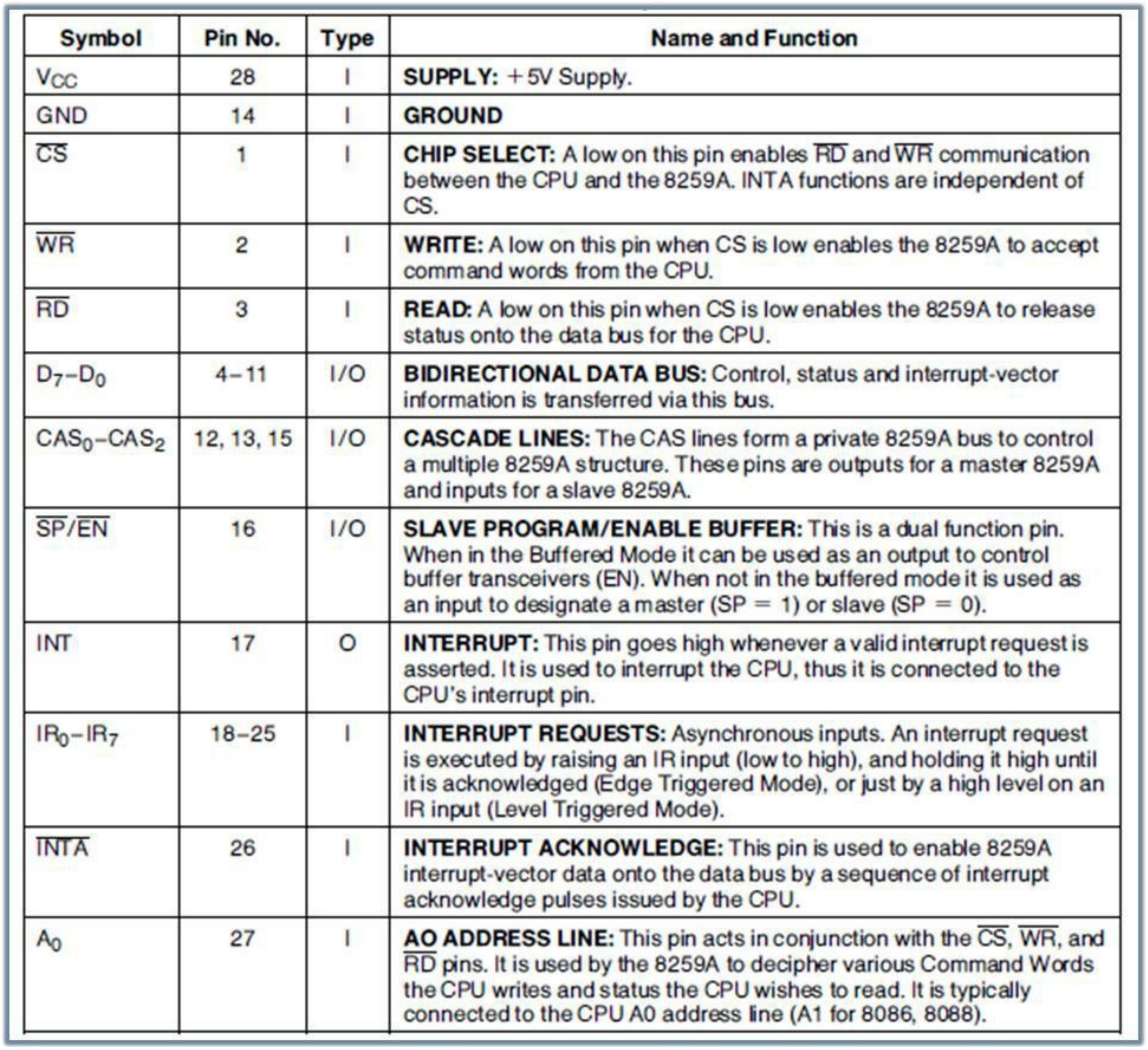


Fig.17 Pin Diagram of 8259

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).

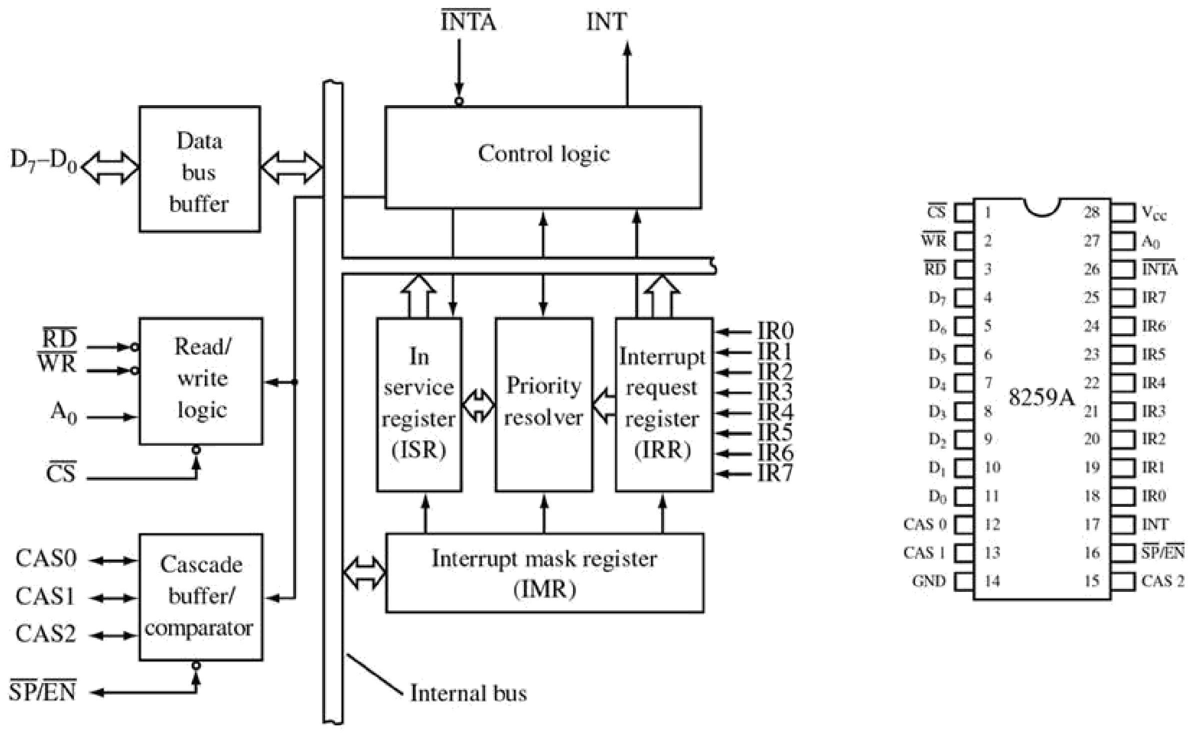


Fig. 18 Block Diagram of 8259

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.

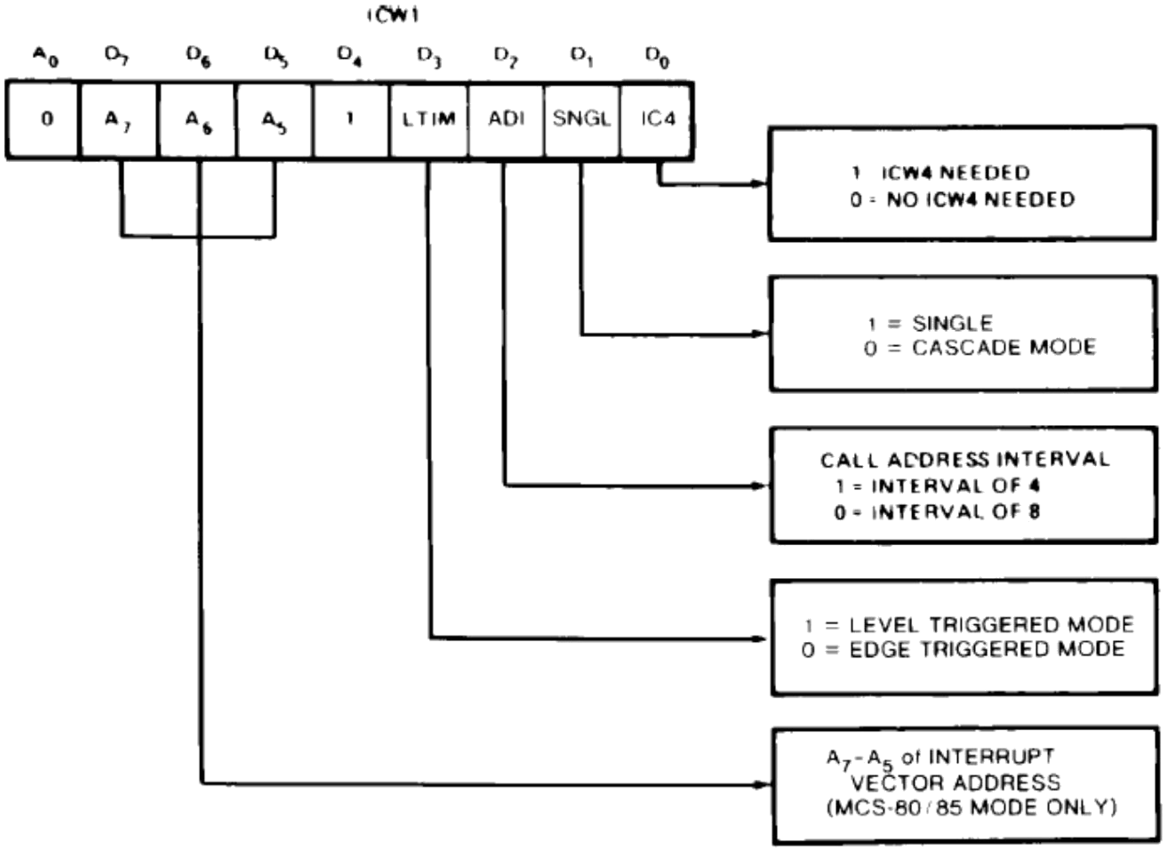
When the 8259A PIC receives an interrupt, INT be-comes active and an interrupt acknowledge cycle is started. If a higher priority interrupt occurs between the two INTA pulses, the INT line goes inactive immediately after the second INTA pulse. After an un-specified amount of time the INT line is activated 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.

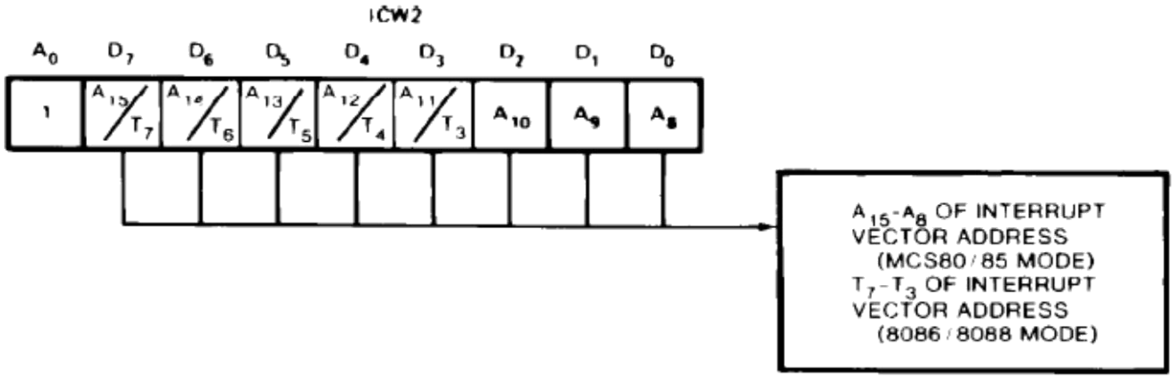
INITIALIZATION COMMAND WORDS

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)

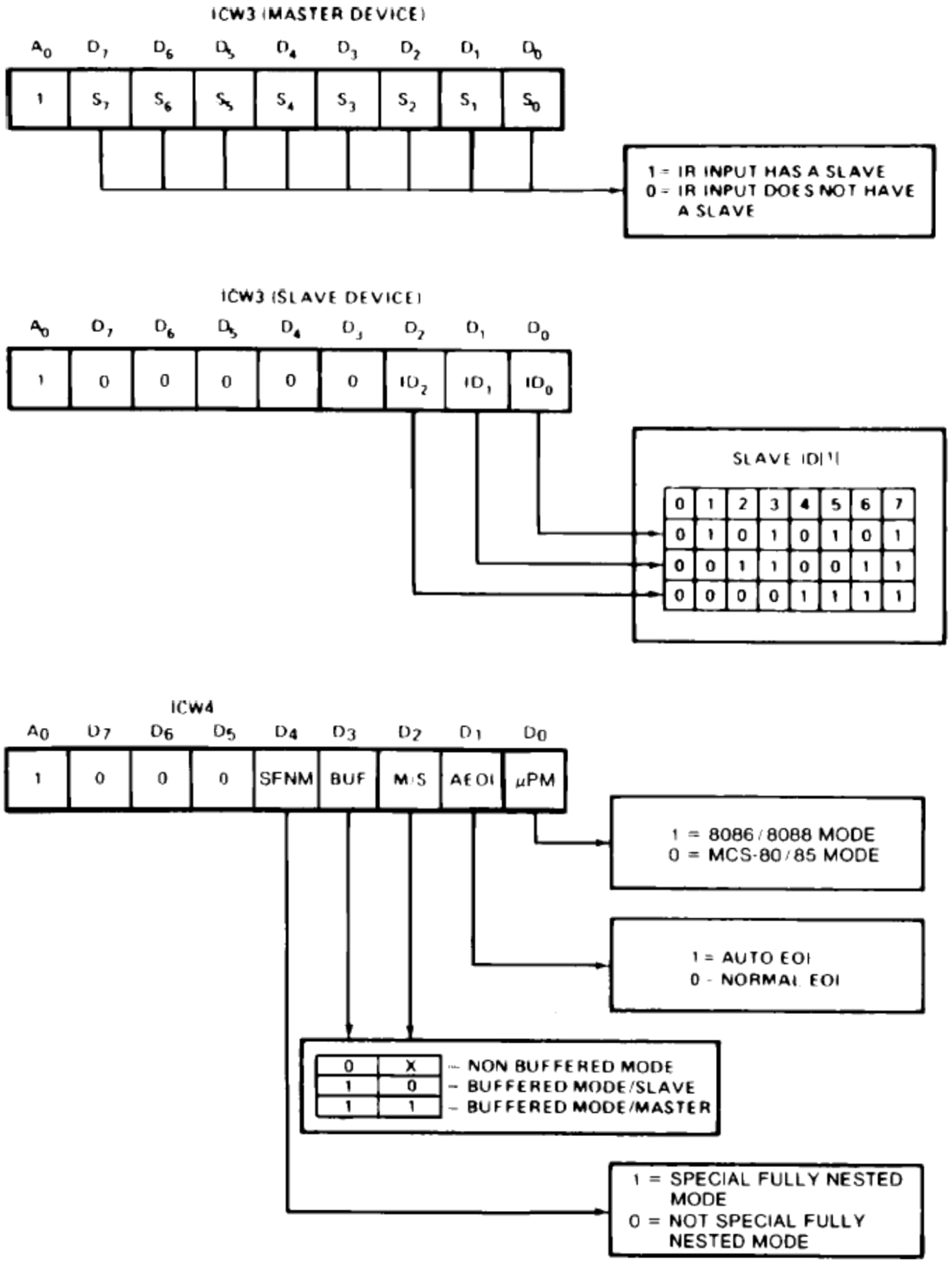

ICW3 (MASTER DEVICE)
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 S7 S6 S5 S4 S3 S2 S1 S0
1 = IR INPUT HAS A SLAVE
0 = IR INPUT DOES NOT HAVE A SLAVE
ICW3 (SLAVE DEVICE)
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 0 0 0 0 0 ID2 ID1 ID0
SLAVE ID(1)
0 1 2 3 4 5 6 7
0 1 0 1 0 1 0 1
0 0 1 1 0 0 1 1
0 0 0 0 1 1 1 1
ICW4
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 0 0 0 SFNM BUF M/S AEOI μPM
1 = 8086/8088 MODE
0 = MCS-80/85 MODE
1 = AUTO EOI
0 - NORMAL EOI
0 X – NON BUFFERED MODE
1 0 – BUFFERED MODE/SLAVE
1 1 – BUFFERED MODE/MASTER
1 = SPECIAL FULLY NESTED MODE
0 = NOT SPECIAL FULLY NESTED MODE

Fig 19 . 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

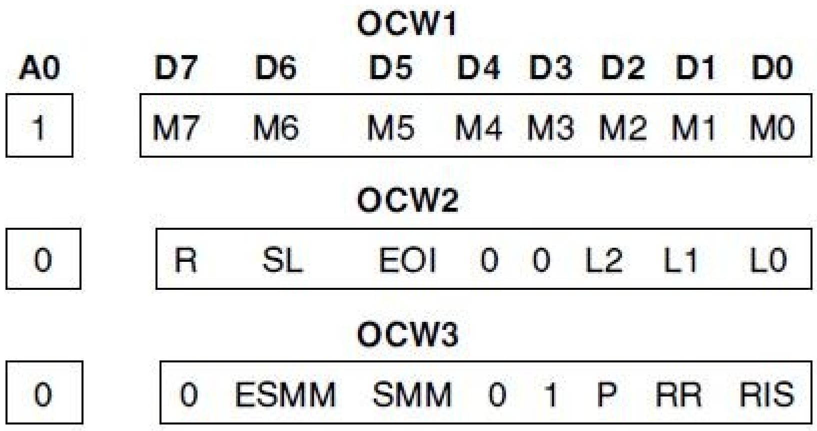
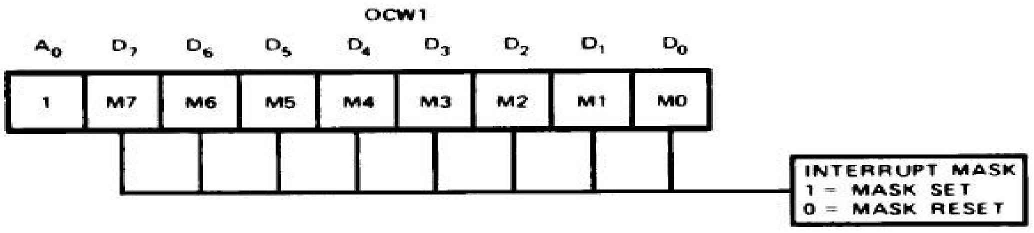


Fig . Operational Control Words



OCW1
A_0 D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
1 M7 M6 M5 M4 M3 M2 M1 M0
INTERRUPT MASK
1 = MASK SET
0 = MASK RESET

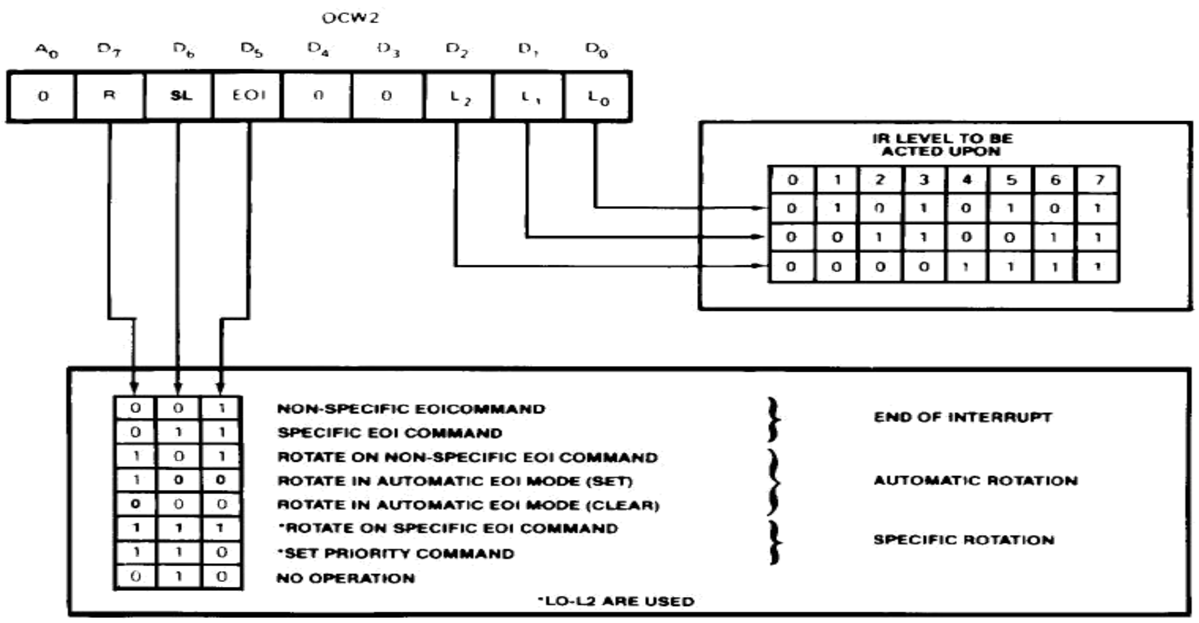

OCW2
A0 D7 D6 D5 D4 D3 D2 D1 D0
0 R SL EOI 0 0 L2 L1 L0
IR LEVEL TO BE ACTED UPON
0 1 2 3 4 5 6 7
0 1 0 1 0 1 0 1
0 0 1 1 0 0 1 1
0 0 0 0 1 1 1 1
0 0 1 NON-SPECIFIC EOICOMMAND
0 1 1 SPECIFIC EOI COMMAND
1 0 1 ROTATE ON NON-SPECIFIC EOI COMMAND
1 0 0 ROTATE IN AUTOMATIC EOI MODE (SET)
0 0 0 ROTATE IN AUTOMATIC EOI MODE (CLEAR)
1 1 1 *ROTATE ON SPECIFIC EOI COMMAND
1 1 0 *SET PRIORITY COMMAND
0 1 0 NO OPERATION
END OF INTERRUPT
AUTOMATIC ROTATION
SPECIFIC ROTATION
*LO-L2 ARE USED

Fig 20 Operation Command Word Format

INTERFACING MEMORY CHIPS WITH 8085

8085 has 16 address lines (A0 - A15), hence a maximum of 64 KB (= 2^{16} bytes) of memory locations can be interfaced with it. The memory address space of the 8085 takes values from 0000H to FFFFH.

The 8085 initiates set of signals such as $IO/\overline{M}$, $\overline{RD}$ and $\overline{WR}$ when it wants to read from and write into memory. Similarly, each memory chip has signals such as CE or CS (chip enable or chip select), $\overline{OE}$ or $\overline{RD}$ (output enable or read) and $\overline{WE}$ or $\overline{WR}$ (write enable or write) associated with it.

Generation of Control Signals for Memory:

When the 8085 wants to read from and write into memory, it activates $IO/\overline{M}$, $\overline{RD}$ and $\overline{WR}$ signals as shown in Table 8.

Table 8 Status of $IO/\overline{M}$, $\overline{RD}$ and $\overline{WR}$ signals during memory read and write operations

$IO/\overline{M}$	$\overline{RD}$	$\overline{WR}$	Operation
0	0	1	8085 reads data from memory
0	1	0	8085 writes data into memory

Using $IO/\overline{M}$, $\overline{RD}$ and $\overline{WR}$ signals, two control signals $\overline{MEMR}$ (memory read) and $\overline{MEMW}$ (memory write) are generated. Fig. 16 shows the circuit used to generate these signals.

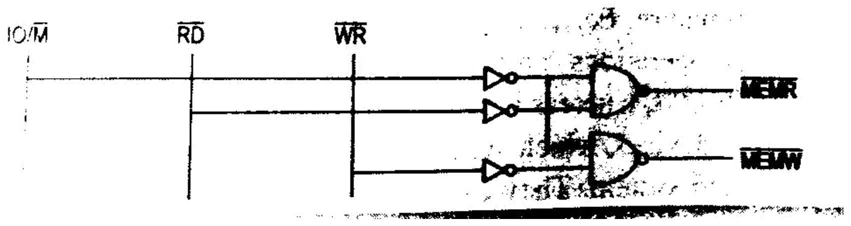


Fig. 16 Circuit used to generate $\overline{MEMR}$ and $\overline{MEMW}$ signals

When is $IO/\overline{M}$ high, both memory control signals are deactivated irrespective of the status of $\overline{RD}$ and $\overline{WR}$ signals.

Ex: Interface an IC 2764 with 8085 using NAND gate address decoder such that the address range allocated to the chip is 0000H – 1FFFH.

Specification of IC 2764:

 8 KB (8 x 2^{10} byte) EPROM chip
13 address lines (2^{13} bytes = 8 KB)

Interfacing:

- 13 address lines of IC are connected to the corresponding address lines of 8085.
- Remaining address lines of 8085 are connected to address decoder formed using logic gates, the output of which is connected to the $\overline{CE}$ pin of IC.
- Address range allocated to the chip is shown in Table 9.
- Chip is enabled whenever the 8085 places an address allocated to EPROM chip in the address bus. This is shown in Fig. 17.

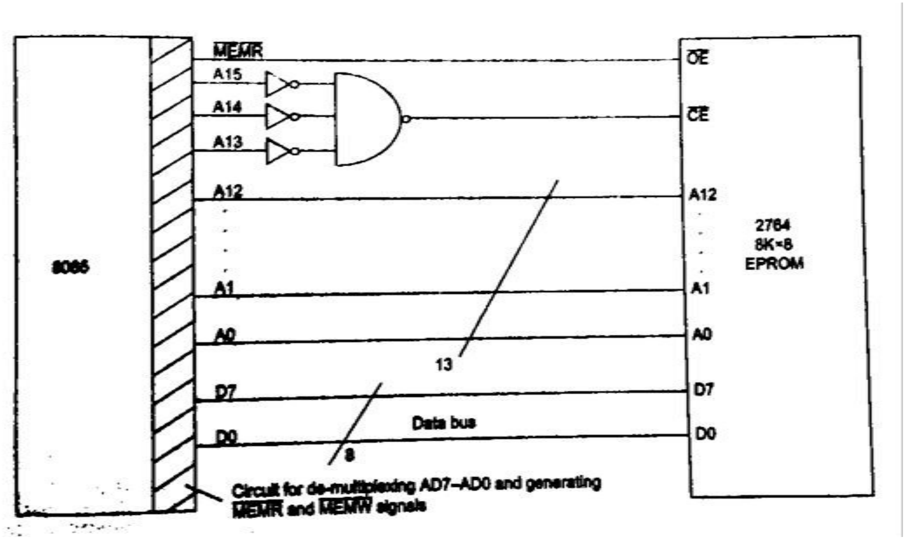


Fig. 17 Interfacing IC 2764 with the 8085

Table 9 Address allocated to IC 2764

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001H
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1FFEH
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH

Ex: Interface a 6264 IC (8K x 8 RAM) with the 8085 using NAND gate decoder such that the starting address assigned to the chip is 4000H.

Specification of IC 6264:

- 8K x 8 RAM
- 8 KB = 2^{13} bytes
- 13 address lines

The ending address of the chip is 5FFFH (since 4000H + 1FFFH = 5FFFH). When the address 4000H to 5FFFH are written in binary form, the values in the lines A15, A14, A13 are 0, 1 and 0 respectively. The NAND gate is designed such that when the lines A15 and A13 carry 0 and A14 carries 1, the output of the NAND gate is 0. The NAND gate output is in turn connected to the CE1 pin of the RAM chip. A NAND output of 0 selects the RAM chip for read or write operation, since CE2 is already 1 because of its connection to +5V. Fig. 18 shows the interfacing of IC 6264 with the 8085.

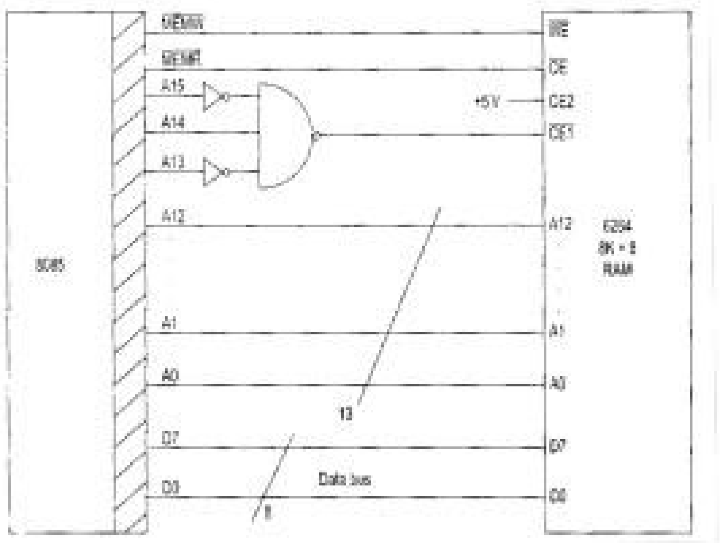


Fig. 18 Interfacing 6264 IC with the 8085

Ex: Interface two 6116 ICs with the 8085 using 74LS138 decoder such that the starting addresses assigned to them are 8000H and 9000H, respectively.

Specification of IC 6116:

- 2 K x 8 RAM
- 2 KB = 2^{11} bytes
- 11 address lines

6116 has 11 address lines and since 2 KB, therefore ending addresses of 6116 chip 1 is and chip 2 are 87FFH and 97FFH, respectively. Table 10 shows the address range of the two chips.

Table 10 Address range for IC 6116

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000H
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	87FFH (RAM chip 1)
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	9000H
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	97FFH (RAM chip 2)

Interfacing:

- Fig. 19 shows the interfacing.
- A0 – A10 lines of 8085 are connected to 11 address lines of the RAM chips.
- Three address lines of 8085 having specific value for a particular RAM are connected to the three select inputs (C, B and A) of 74LS138 decoder.
- Table 10 shows that A13=A12=A11=0 for the address assigned to RAM 1 and A13=0, A12=1 and A11=0 for the address assigned to RAM 2.
- Remaining lines of 8085 which are constant for the address range assigned to the two RAM are connected to the enable inputs of decoder.
- When 8085 places any address between 8000H and 87FFH in the address bus, the select inputs C, B and A of the decoder are all 0. The Y0 output of the decoder is also 0, selecting RAM 1.
- When 8085 places any address between 9000H and 97FFH in the address bus, the select inputs C, B and A of the decoder are 0, 1 and 0. The Y2 output of the decoder is also 0, selecting RAM 2.

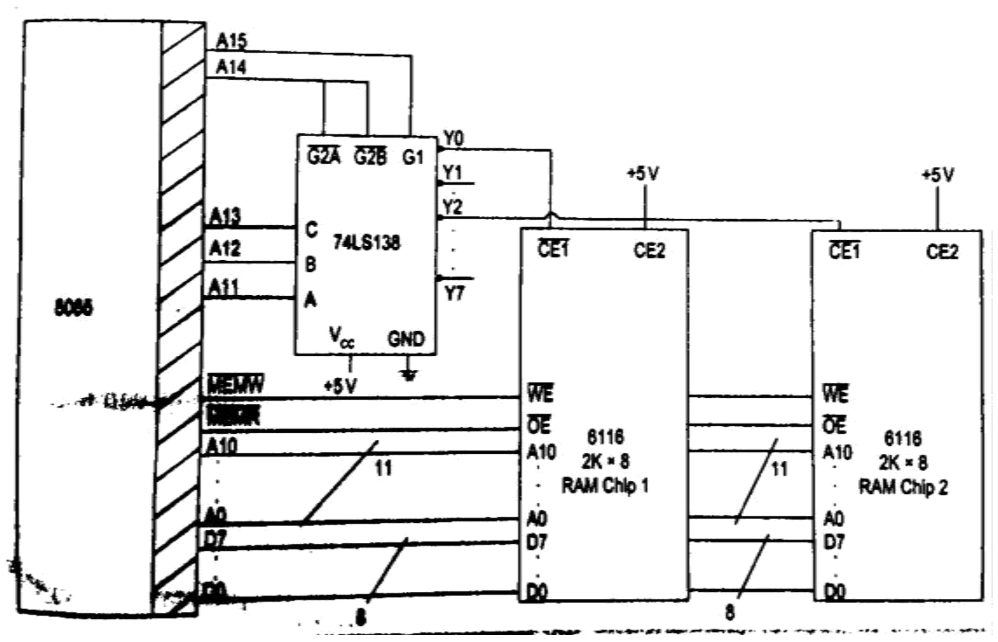


Fig. 19 Interfacing two 6116 RAM chips using 74LS138 decoder

3. **PERIPHERAL MAPPED I/O INTERFACING**

In this method, the I/O devices are treated differently from memory chips. The control signals I/O read ($\overline{IOR}$) and I/O write ($\overline{IOW}$), which are derived from the IO/$\overline{M}$, $\overline{RD}$ and $\overline{WR}$ signals of the 8085, are used to activate input and output devices, respectively.

Generation of these control signals is shown in Fig. 20. Table 11 shows the status of IO/$\overline{M}$, $\overline{RD}$ and $\overline{WR}$ signals during I/O read and I/O write operation.

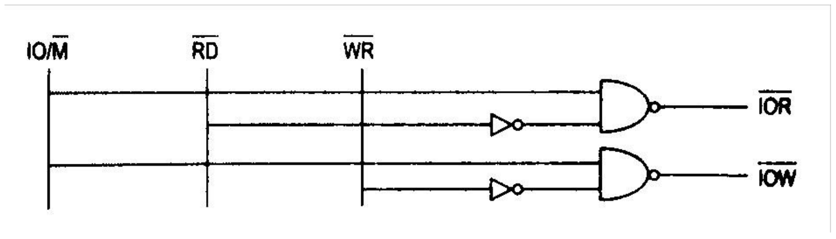


Fig. 20 Generation of $\overline{IOR}$ and $\overline{IOW}$ signals

IN instruction is used to access input device and OUT instruction is used to access output device. Each I/O device is identified by a unique 8-bit address assigned to it. Since the control signals used to access input and output devices are different, and all I/O device use 8-bit address, a maximum of 256 (2^8) input devices and 256 output devices can be interfaced with 8085.

Table 11 Status of $\overline{IOR}$ and $\overline{IOW}$ signals in 8085.

IO/$\overline{M}$	$\overline{RD}$	$\overline{WR}$	$\overline{IOR}$	$\overline{IOW}$	Operation
1	0	1	0	1	I/O read operation
1	1	0	1	0	I/O write operation
0	X	X	1	1	Memory read or write operation

Ex: Interface an 8-bit DIP switch with the 8085 such that the address assigned to the DIP switch if F0H.

IN instruction is used to get data from DIP switch and store it in accumulator. Steps involved in the execution of this instruction are:

- Address F0H is placed in the lines A0 – A7 and a copy of it in lines A8 – A15.

ii. The $\overline{\text{IOR}}$ signal is activated ($\overline{\text{IOR}}$ = 0), which makes the selected input device to place its data in the data bus.

iii. The data in the data bus is read and store in the accumulator.

Fig. 21 shows the interfacing of DIP switch.

A7	A6	A5	A4	A3	A2	A1	A0	
1	1	1	1	0	0	0	0	= F0H

A0 – A7 lines are connected to a NAND gate decoder such that the output of NAND gate is 0. The output of NAND gate is ORed with the $\overline{\text{IOR}}$ signal and the output of OR gate is connected to $\overline{\text{1G}}$ and $\overline{\text{2G}}$ of the 74LS244. When 74LS244 is enabled, data from the DIP switch is placed on the data bus of the 8085. The 8085 read data and store in the accumulator. Thus data from DIP switch is transferred to the accumulator.

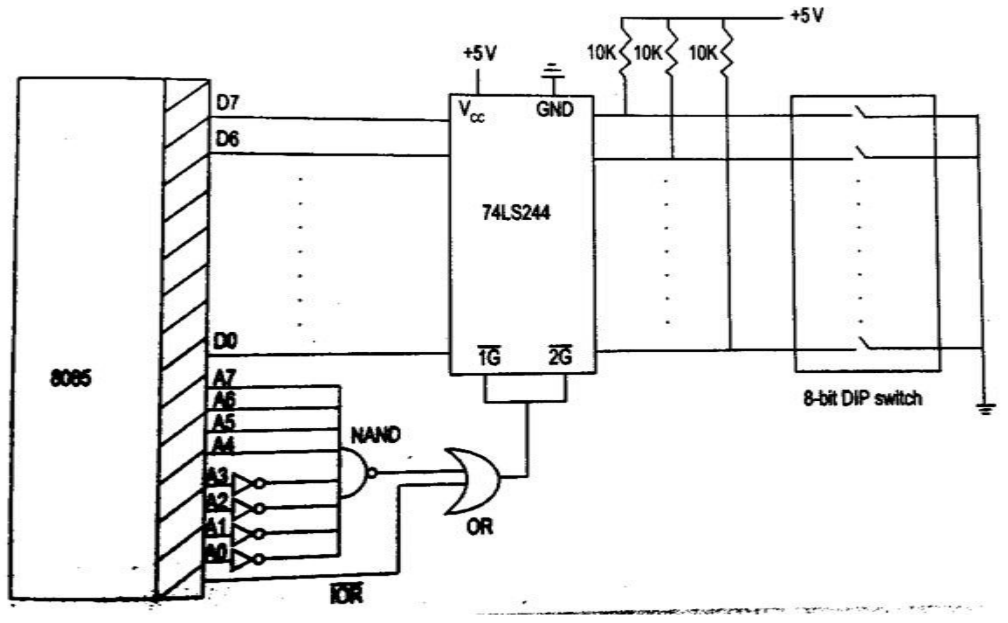


Fig. 21 interfacing of 8-bit DIP switch with 8085

4. **MEMORY MAPPED I/O INTERFACING**

In memory-mapped I/O, each input or output device is treated as if it is a memory location.

The $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ control signals are used to activate the devices. Each input or output device is identified by unique 16-bit address, similar to 16-bit address assigned to memory location. All memory related instruction like LDA 2000H, LDAX B, MOV A, M can be used.

Since the I/O devices use some of the memory address space of 8085, the maximum memory capacity is lesser than 64 KB in this method.

Ex: Interface an 8-bit DIP switch with the 8085 using logic gates such that the address assigned to it is F0F0H.

Since a 16-bit address has to be assigned to a DIP switch, the memory-mapped I/O technique must be used. Using LDA F0F0H instruction, the data from the 8-bit DIP switch can be transferred to the accumulator. The steps involved are:

- The address F0F0H is placed in the address bus A0 – A15.
- The $\overline{\text{MEMR}}$ signal is made low for some time.
- The data in the data bus is read and stored in the accumulator.

Fig. 22 shows the interfacing diagram.

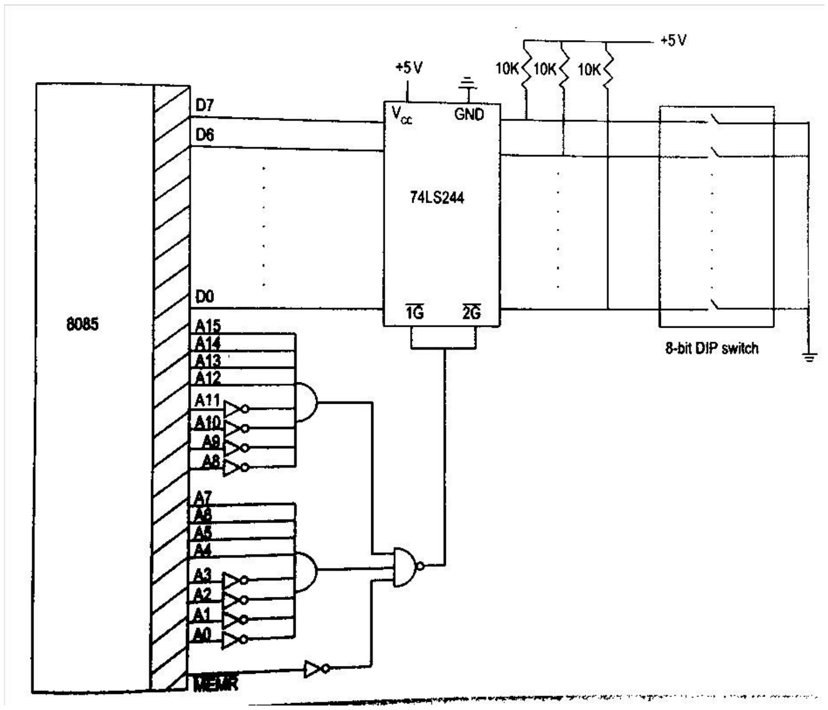


Fig. 22 Interfacing 8-bit DIP switch with 8085

When 8085 executes the instruction LDA F0F0H, it places the address F0F0H in the address lines A0 – A15 as:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	= F0F0H

The address lines are connected to AND gates. The output of these gates along with MEMR signal are connected to a NAND gate, so that when the address F0F0H is placed in the

address bus and MEMR = 0 its output becomes 0, thereby enabling the buffer 74LS244. The data from the DIP switch is placed in the 8085 data bus. The 8085 reads the data from the data bus and stores it in the accumulator.

UNIT 4 INTRODUCTION TO MICROCONTROLLER 9 Hrs.
Introduction to microcontrollers, Difference between microprocessors and microcontrollers, Architectural features of 8051, I/O Ports, Interrupts, Addressing Modes and Instruction set of 8051, Programming examples.

INTEL 8051 MICROCONTROLLER

- WHAT IS A MICROCONTROLLER?
    - **All of the components needed for a controller were built right onto one chip.**
    - **A one chip computer, or microcontroller was born.**
    - **A microcontroller is a highly integrated chip which includes, on one chip, all or most of the parts needed for a controller.**
    - **The microcontroller could be called a "one-chip solution".**

- **8051 Family**

The 8051 is just one of the MCS-51 family of microcontrollers developed by Intel. The design of each of the MCS-51 microcontrollers are more or less the same. The differences between each member of the family is the amount of on-chip memory and the number of timers, as detailed in the table below.

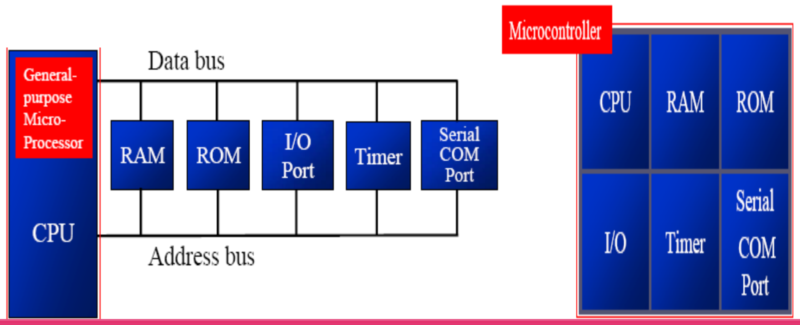
Microcontroller	**On-chip Code Memory**	**On-chip Data Memory**	**Timers**
8051	4K ROM	128 bytes	2
8031	0	128 bytes	2
8751	4K EPROM	128 bytes	2
8052	8K ROM	256 bytes	3
8032	0	256 bytes	3
8752	8K EPROM	256 bytes	3

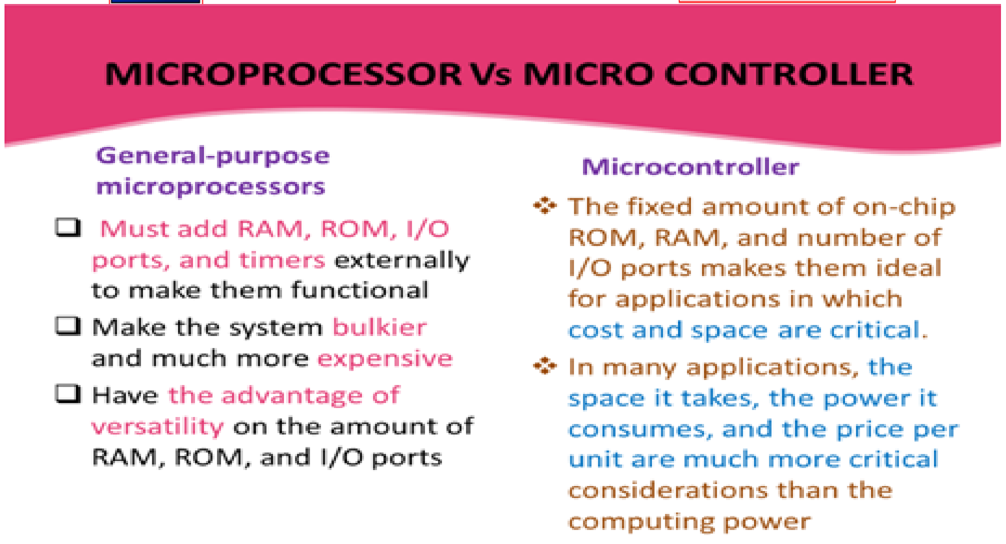
Each chip also contains:

- four 8-bit input/output (I/0) ports
- serial interface
- 64K external code memory space

- 64K external data memory space
- Boolean processor
- 210 bit-addressable locations
- 4us multiply/divide

1.2 MICROPROCESSOR vs MICRO CONTROLLER





MICROPROCESSOR Vs MICRO CONTROLLER

General-purpose microprocessors

- ❑ Must add RAM, ROM, I/O ports, and timers externally to make them functional
- ❑ Make the system bulkier and much more expensive
- ❑ Have the advantage of versatility on the amount of RAM, ROM, and I/O ports

Microcontroller

- ❖ The fixed amount of on-chip ROM, RAM, and number of I/O ports makes them ideal for applications in which cost and space are critical.
- ❖ In many applications, the space it takes, the power it consumes, and the price per unit are much more critical considerations than the computing power

Disadvantages of microprocessor

- The overall system cost is high
- A large sized PCB is required for assembling all the components
- Overall product design requires more time
- Physical size of the product is big
- A discrete components are used, the system is not reliable

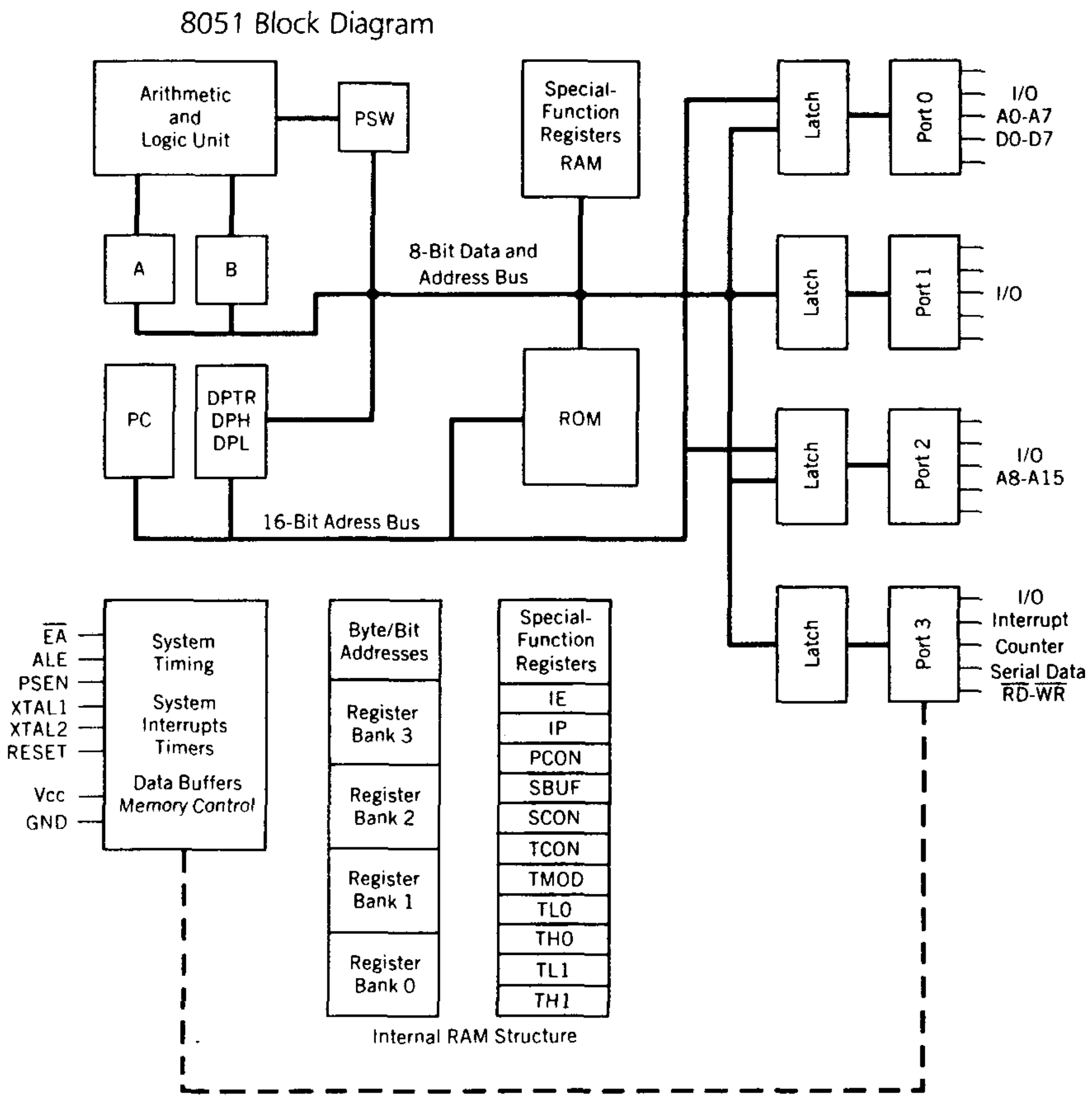
Advantages of Microcontroller based System

- As the peripherals are integrated into a single chip, the overall system cost is very less
- The product is of small size compared to microprocessor based system
- The system design now requires very little efforts

- As the peripherals are integrated with a microprocessor the system is more reliable
- Though microcontroller may have on chip ROM,RAM and I/O ports, addition ROM, RAM I/O ports may be interfaced externally if required
- On chip ROM provide a software security

1.3 8051 Block Diagram

8051 Internal Architecture



8051 CPU Registers

- **ACCUMULATOR (ACC)**
    - Operandregister
    - Implicit or specified in the instruction
    - Has an address in on chip SFR bank
- **B REGISTER**
    - to store one of the operands for multiplication and division
    - otherwise, scratch pad
    - considered as a SFR
- **PROGRAM STATUS WORD (PSW)**
    - Set of flags contain status information
    - One of the SFR
- **STACK POINTER (SP)**
    - 8 bit wide register
    - Incremented before data is stored on to the stack using PUSH or CALL instructions
    - Stack defined anywhere on the 128 byte RAM
    - RESET → initiated to 0007H
    - Not a top to down structure
    - Allotted an address in SFR
- **DATA POINTER (DPTR)**
    - 16 bit register
    - contains DPH and DPL
    - Pointer to external RAM address
    - DPH and DPL allotted separate addresses in SFR bank
- **PORT 0 TO 3 LATCHES & DRIVERS**
    - Each i/o port allotted a latch and a driver
    - Latches allotted address in SFR
    - User can communicate via these ports
    - P0, P1, P2,P3
- **SERIAL DATA BUFFER**
    - internally had TWO independent registers
    - TRANSMIT buffer →parallel in serial out (PISO)
    - RECEIVE buffer → serial in parallel out (SIPO)
    - identified by SBUF and allotted an address in SFR

- byte written to SBUF → initiates serial TX
    - byte read from SBUF → reads serially received data

- **TIMER REGISTERS**
    - for Timer0 (16 bit register – TL0 & TH0)
    - for Timer1 (16 bit register – TL1 & TH1)
    - four addresses allotted in SFR
- **CONTROL REGISTERS**
    - IP
    - IE
    - TMOD
    - TCON
    - SCON
    - PCON
    - contain control and status information for interrupts, timers/counters and serial port
    - Allotted separate address in SFR
- **TIMING AND CONTROL UNIT**
    - derives necessary timing and control signals

For internal circuit and external system bus

- **OSCILLATOR**
    - generates basic timing clock signal using crystal oscillator
- **INSTRUCTION REGISTER**
    - decodes the opcode and gives information to timing and control unit
- **EPROM & PROGRAM ADDRESS REGISTER**
    - provide on chip EPROM and mechanism to address it
    - All versions don't have EPROM
- **RAM & RAM ADDRESS REGISTER**
    - provide internal 128 bytes RAM and a mechanism to address internally
- **ALU**
    - Performs 8 bit arithmetic and logical operations over the operands held by TEMP1 and TEMP 2
    - User cannot access temporary registers

- **SFR REGISTER BANK**
  - set of special function registers
  - address range : 80 H to FF H
    - Interrupt, serial port and timer units control and perform specific functions under the control of timing and control unit

1.4 Addressing Modes

The five addressing modes are:

- Immediate
- Register
- Direct
- Indirect
- Indexed

Immediate Addressing

If the operand is a constant then it can be stored in memory immediately after the opcode. Remember, values in code memory (ROM) do not change once the system has been programmed and is in use in the everyday world. Therefore, immediate addressing is only of use when the data to be read is a constant.

For example, if your program needed to perform some calculations based on the number of weeks in the year, you could use immediate addressing to load the number 52 (34H) into a register and then perform arithmetic operations upon this data.

```
MOV R0, #34
```

The above instruction is an example of immediate addressing. It moves the data 34H into R0. The assembler must be able to tell the difference between an address and a piece of data. The has symbol (#) is used for this purpose (whenever the assembler sees # before a number it knows this is immediate addressing). This is a two-byte instruction.

Register Addressing

Often we need to move data from a register into the accumulator so that we can perform arithmetic operations upon it. For example, we may wish to move the contents of R5 into the accumulator.

```
MOV A, R5
```

This is an example of register addressing. It moves data from R5 (in the currently selected register bank) into the accumulator.

ADD A, R6

The above is another example of register addressing. It adds the contents of R6 to the accumulator, storing the result in the accumulator. Note that in both examples the destination comes first. This is true of all instructions.

Direct Addressing

Direct addressing is used for accessing data in the on-chip RAM. Since there are 256 bytes of RAM (128 bytes general storage for the programmer and another 128 bytes for the SFRs). That means the addresses go from 00H to FFH, any of which can be stored in an 8-bit location.

MOV A, 67

The above instruction moves the data in location 67H into the accumulator. Note the difference between this and immediate addressing. Immediate addressing uses the data, which is immediately after the instruction. With direct addressing, the operand is an address. The data to be operated upon is stored in that address. The assembler realises this is an address and not data because there is no hash symbol before it.

ADD A, 06

The above instruction adds the contents of location 06H to the accumulator and stores the result in the accumulator. If the selected register bank is bank 0 then this instruction is the same as ADD A, R6.

Indirect Addressing

Register addressing and direct addressing both restrict the programmer to manipulation of data in fixed addresses. The address the instruction reads from (MOV A, 30H) or writes to (MOV 30H, A) cannot be altered while the program is running. There are times when it is necessary to read and write to a number of contiguous memory locations. For example, if you had an array of 8-bit numbers stored in memory, starting at address 30H, you may wish to examine the contents of each number in the array (perhaps to find the smallest number). To do so, you would need to read location 30H, then 31H, then 32H and so on. This can be achieved using indirect addressing. R0 and R1 may be used as pointer registers. We can use either one to store the current memory location and then use the indirect addressing instruction shown below.

MOV A, @Ri

whereRi is either R0 or R1. Now, we can read the contents of location 30H through indirect addressing:

MOV R0, #30H
MOV A, @R0

The first instruction is an example of immediate addressing whereby the data 30H is placed in R0. The second instruction is indirect addressing. It moves the contents of location 30H into the accumulator.

If we now wish to get the data in location 31H we use the following:

INC R0
MOV A, @R0

Once we see how to write a loop in assembly language, we will be able to read the entire contents of the array.

Index Addressing Mode & On-chip ROM Access

- Limitation of register indirect addressing:
- 8-bit addresses (internal RAM)
- DPTR: 16 bits
- MOVC A, @A+DPTR ; "C" means program (code) space ROM

1.5 Special Function Registers (SFRs)

Locations 80H to FFH contain the special function registers. As you can see from the diagram above, not all locations are used by the 8051 (eleven locations are blank). These extra locations are used by other family members (8052, etc.) for the extra features these microcontrollers possess. Also note that not all SFRs are bit-addressable. Those that are have a unique address for each bit. We will deal with each of the SFRs as we progress through the course, but for the moment you should take note of the accumulator (ACC) at address E0H and the four port registers at addresses 80H for P0, 90h for P1, A0 for P2 and B0 for P3. We will later see how easy this makes ready from and writing to any of the four ports. **Program Status Word (PSW)**

CY	AC	F0	RS1	RS0	OV		P

The PSW is at location D0H and is bit addressable. The table below describes the function of each bit.

Bit	**Symbol**	**Address**	**Description**
PSW.7	CY	D7H	Carry flag
PSW.6	AC	D6H	Auxiliary carry flag
PSW.5	F0	D5H	Flag 0
PSW.4	RS1	D4H	Register bank select 1
PSW.3	RS0	D3H	Register bank select 0
PSW.2	OV	D2H	Overflow flag
PSW.1	--	D1H	Reserved
PSW.0	P	D0H	Even parity flag

Carry Flag The carry flag has two functions.

- Firstly, it is used as the carry-out in 8-bit addition/subtraction. For example, if the accumulator contains FDH and we add 3 to the contents of the accumulator (ADD A, #3), the accumulator will then contain zero and the carry flag will be set. It is also set if a subtraction causes a borrow into bit 7. In other words, if a number is subtracted from another number smaller than it, the carry flag will be set. For example, if A contains 3DH and R3 contains 4BH, the instruction SUBB A, R3 will result in the carry bit being set (4BH is greater than 3DH).
- The carry flag is also used during Boolean operations. For example, we could AND the contents of bit 3DH with the carry flag, the result being placed in the carry flag - ANL C, 3DH

Register Bank Select Bits Bits 3 and 4 of the PSW are used for selecting the register bank. Since there are four register banks, two bits are required for selecting a bank, as detailed below.

PSW.4	PSW.3	Register Bank	Address of Register Bank
0	0	0	00H to 07H
0	1	1	08H to 0FH
1	0	2	10H to 17H
1	1	3	18H to 1FH

For example, if we wished to activate register bank 3 we would use the following instructions -

```
SETB RS1
SETB RS0
```

If we then moved the contents of R4 to the accumulator (MOV A, R4) we would be moving the data from location
1CH to A.

Flag 0

Flag 0 is a general-purpose flag available to the programmer.

Parity Bit

The parity bit is automatically set or cleared every machine cycle to ensure even parity with the accumulator. The number of 1-bits in the accumulator plus the parity bit is always even. In other words, if the number of 1s in the accumulator is odd then the parity bit is set to make the overall number of bits even. If the number of 1s in the accumulator is even then the parity bit is cleared to make the overall number of bits even. For example, if the accumulator holds the number 05H, this is 0000 0101 in binary => the accumulator has an even number of 1s, therefore the parity bit is cleared. If the accumulator holds the number F2H, this is 1111 0010 => the accumulator has an odd number of 1s, therefore the parity bit is set to make the overall number of 1s even. As we shall see later in the course, the parity bit is most often used for detecting errors in transmitted data.

B Register

The B register is used together with the accumulator for multiply and divide operations.

- The MUL AB instruction multiplies the values in A and B and stores the low-byte of the result in A and the high-byte in B.
    - For example, if the accumulator contains F5H and the B register contains 02H, the result of MUL AB will be A = EAH and B = 01H.
- The DIV AB instruction divides A by B leaving the integer result in A and the remainder by B.

- For example, if the accumulator contains 07H and the B register contains 02H, the result of DIV AB will be A = 03H and B = 01H.

The B register is also bit-addressable.

Stack Pointer

The stack pointer (SP) is an 8-bit register at location 81H. A stack is used for temporarily storing data. It operates on the basis of last in first out (LIFO). Putting data onto the stack is called "pushing onto the stack" while taking data off the stack is called "popping the stack." The stack pointer contains the address of the item currently on top of the stack. On power-up or reset the SP is set to 07H. When pushing data onto the stack, the SP is first increased by one and the data is then placed in the location pointed to by the SP. When popping the stack, the data is taken off the stack and the SP is then decreased by one. Since reset initialises the SP to 07H, the first item pushed onto the stack is stored at 08H (remember, the SP is incremented first, then the item is placed on the stack). However, if the programmer wishes to use the register banks 1 to 3, which start at address 08H, he/she must move the stack to another part of memory. The general purpose RAM starting at address 30H is a good spot to place the stack. To do so we need to change the contents of the SP.

```
MOV SP, #2FH.
```

Now, the first item to be pushed onto the stack will be stored at 30H.

Subroutines and the Stack

We have already looked at calling a subroutine in the previous section. Now we will look at the actual call instructions and the effect they have on the stack.

ACALL and LCALL

ACALL stands for absolute call while LCALL stands for long call. These two instructions allow the programmer to call a subroutine. There is a slight difference between the two instructions, the same as the difference between AJMP and LJMP. ACALL allows you to jump to a subroutine within the same 2K page while LCALL allows you to jump to a subroutine anywhere in the 64K code space. The advantage of ACALL over LCALL is that it is a 2-byte instruction while LCALL is a 3-byte instruction.

Help from the Assembler

In the same way that you, the programmer, may use the assembler instruction JMP anytime you need an unconditional jump and the assembler will replace this with the appropriate 8051 jump instruction (SJMP, AJMP or LJMP), you may use the assembler CALL instruction and it will be replaced by the appropriate 8051 subroutine call instruction (ACALL or LCALL - note there is no SCALL instruction).

LCALL Operation

Since the ACALL and LCALL instructions perform almost the same functions we will look at the operation of the LCALL instruction only.

LCALL add16 - long call to subroutine

Encoding - 0001 0010 aaaaaaaaaaaaaaaa (3-byte instruction)

Operation -

(PC) <- (PC) + 3

(SP) <- (SP) + 1

((SP)) <- (PC7-PC0)

(SP) <- (SP) + 1

((SP)) <- (PC15 - PC8)

(PC) <- add15 - add0

The operation is as follows. The PC is increased by 3 (because this is a 3-byte instruction). The stack pointer is incremented so that it points to the next empty space on the stack. The third line reads: the contents of the contents of the SP get the low byte of the PC. On system reset the SP is initialised with the value 07H. Therefore, the first item pushed onto the stack will be stored in location 08H. Therefore:

((SP)) is equivalent to (08) - meaning location 08H in memory gets the low bye of the PC - ((SP)) <- (PC7 - PC0)

After the low byte of the PC has been stored on the stack the SP is incremented to point to the next empty space on the stack (ie; the SP now contains 09H).

SP)) is now equivalent to (09) - meaning location 09H in memory gets the high bye of the PC - ((SP)) <- (PC15 - PC8)

Now that the PC has been stored on the stack the PC is loaded with the 16-bit address (add15 - add0). Subroutines are generally sections of code that will be used many times by the system. A subroutine might be used for taking information from a keyboard or writing data to a serial link. A particular subroutine will be stored at some point in code memory, but it can be called from any location in the program. Therefore, the system needs some way of knowing where to jump back to once execution of the subroutine is complete. The first diagram below shows the contents of the PC and the SP as the instruction LCALL sub (at location 103BH in code memory) is about to be executed. Notice the SP is at its reset value of 07H and the PC contains the address of the next instruction to be executed

RET

A subroutine must end with the *RET* instruction, which simply means *return from subroutine*. Since a subroutine can be called from anywhere in code memory, the *RET* instruction does not specify where to return to. The return address may be different each time the subroutine is called. If you take the flashing LED program from the last section, we called *twoLoopDelay* in the main program after we had turned on the LED. But we also called *twoLoopDelay* from within *threeLoopDelay*. In these two calls the return address is different; in the first call we are returning to the main program once *twoLoopDelay* has completed, but on the second call we are returning to *threeLoopDelay*.

The system knows where to return to because, as we have seen above, the return address (ie; the address of the next instruction after the *LCALL*) is stored on the stack. Therefore, the operation of the *RET* instruction is:

RET - return from subroutine

Encoding - 0010 0010

Operation -

(PC15 - PC8) <- ((SP))

(SP) <- (SP) - 1

(PC7 - PC0) <- ((SP))

(SP) <- (SP) - 1

If you look at the diagram above, note the SP contains *09H* - it's pointing at the high-byte of the return address. Therefore, the contents of location *09H* are placed in the high-byte of the PC (PC15 - PC8).

The SP is then decremented (it now contains *08H*) so that it points at the low-byte of the return address. So, the contents of location *08H* are placed in the low-byte of the PC (PC7 - PC0).

In our example above, the PC will now contain *103EH*, and execution takes up immediately after the *LCALL sub* instruction. Also note that the stack is now empty - SP contains 07H.

II INSTRUCTION SET (8051)

2.1. Introduction

The process of writing program for the microcontroller mainly consists of giving instructions (commands) in the specific order in which they should be executed in order to carry out a specific task. As electronics cannot "understand" what for example an instruction "if the push button is pressed- turn the light on" means, then a certain number of simpler and precisely defined orders that decoder can recognise must be used. All commands are known as INSTRUCTION SET.

All microcontrollers compatibile with the 8051 have in total of 255 instructions, i.e. 255 different words available for program writing. At first sight, it is imposing number of odd signs that must be known by heart. However, It is not so complicated as it looks like. Many instructions are considered to be "different", even though they perform the same operation, so there are only 111 truly different commands.

For example: ADD A,R0, ADD A,R1, ... ADD A,R7 are instructions that perform the same operation (additon of the accumulator and register). Since there are 8 such registers, each instruction is counted separately. Taking into account that all instructions perform only 53 operations (addition, subtraction, copy etc.) and most of them are rarely used in practice, there are actually 20-30 abbreviations to be learned, which is acceptable.

2.2 Types of instructions

Depending on operation they perform, all instructions are divided in several groups:

- Arithmetic Instructions
- Branch Instructions
- Data Transfer Instructions
- Logic Instructions
- Bit-oriented Instructions

The first part of each instruction, called MNEMONIC refers to the operation an instruction performs (copy, addition, logic operation etc.). Mnemonics are abbreviations of the name of operation being executed. For example:

- INC R1 - Means: Increment register R1 (increment register R1);
- JNZ LOOP - Means: Jump if Not Zero LOOP (if the number in the accumulator is not 0, jump to the address marked as LOOP);

The other part of instruction, called OPERAND is separated from mnemonic by at least one whitespace and defines data being processed by instructions. Some of the instructions have no operand, while some of them have one, two or three. If there is more than one operand in an instruction, they are separated by a comma. For example:

- RET - return from a subroutine;
- JZ TEMP - if the number in the accumulator is not 0, jump to the address marked as TEMP;
- ADD A, R3 - add R3 and accumulator;
- CJNE A, #20,LOOP - compare accumulator with 20. If they are not equal, jump to the address marked as LOOP;

2.2.1 Arithmetic instructions

Arithmetic instructions perform several basic operations such as addition, subtraction, division, multiplication etc. After execution, the result is stored in the first operand.
For example:
ADD A,R1 - The result of addition (A+R1) will be stored in the accumulator.

ADD A,Rn Adds the register to the accumulator
ADD A,direct Adds the direct byte to the accumulator
ADD A,@Ri Adds the indirect RAM to the accumulator
ADD A,#data Adds the immediate data to the accumulator

ADDC A,Rn Adds the register to the accumulator with a carry flag

ADDC A,direct Adds the direct byte to the accumulator with a carry flag

ADDC A,@Ri Adds the indirect RAM to the accumulator with a carry flag

ADDC A,#data Adds the immediate data to the accumulator with a carry flag

SUBB A,Rn Subtracts the register from the accumulator with a borrow

SUBB A,direct Subtracts the direct byte from the accumulator with a borrow

SUBB A,@Ri Subtracts the indirect RAM from the accumulator with a borrow

SUBB A,#data Subtracts the immediate data from the accumulator with a borrow

INC A Increments the accumulator by 1

INC Rn Increments the register by 1

INC Rx Increments the direct byte by 1

INC @Ri Increments the indirect RAM by 1

DEC A Decrements the accumulator by 1

DEC Rn Decrements the register by 1

DEC Rx Decrements the direct byte by 1

DEC @Ri Decrements the indirect RAM by 1

INC DPTR Increments the Data Pointer by 1

MUL AB Multiplies A and B 1

DIV AB Divides A by B 1

DA A Decimal adjustment of the accumulator according to BCD code

2.2.2 Branch Instructions

There are two kinds of branch instructions:

- Unconditional jump instructions: upon their execution a jump to a new location from where the program continues execution is executed.
- Conditional jump instructions: a jump to a new program location is executed only if a specified condition is met.

Otherwise, the program normally proceeds with the next instruction.

ACALL addr11 Absolute subroutine call

LCALL addr16 Long subroutine call

RET Returns from subroutine

RETI Returns from interrupt subroutine

AJMP addr11 Absolute jump

LJMP addr16 Long jump

SJMP rel Short jump (from –128 to +127 locations relative to the following instruction)

JC rel Jump if carry flag is set. Short jump.

JNC rel Jump if carry flag is not set. Short jump.

JB bit,rel Jump if direct bit is set. Short jump.

JBC bit,rel Jump if direct bit is set and clears bit. Short jump.

JMP @A+DPTR Jump indirect relative to the DPTR

JZ rel Jump if the accumulator is zero. Short jump.

JNZ rel Jump if the accumulator is not zero. Short jump.

CJNE A,direct,rel Compares direct byte to the accumulator and jumps if not equal. Short jump.

CJNE A,#data,rel Compares immediate data to the accumulator and jumps if not equal. Short jump.

CJNE Rn,#data,rel Compares immediate data to the register and jumps if not equal. Short jump.

CJNE @Ri,#data,rel Compares immediate data to indirect register and jumps if not equal. Short jump.

DJNZ Rn,rel Decrements register and jumps if not 0. Short jump.

DJNZ Rx,rel Decrements direct byte and jump if not 0. Short jump.

NOP No operation

2.2.3 Data Transfer Instructions

Data transfer instructions move the content of one register to another. The register the content of which is moved remains unchanged. If they have the suffix "X" (MOVX), the data is exchanged with external memory.

MOV A,Rn Moves the register to the accumulator

MOV A,direct Moves the direct byte to the accumulator

MOV A,@Ri Moves the indirect RAM to the accumulator

MOV A,#data Moves the immediate data to the accumulator

MOV Rn,A Moves the accumulator to the register

MOV Rn,direct Moves the direct byte to the register

MOV Rn,#data Moves the immediate data to the register

MOV direct,A Moves the accumulator to the direct byte

MOV direct,Rn Moves the register to the direct byte

MOV direct,direct Moves the direct byte to the direct byte

MOV direct,@Ri Moves the indirect RAM to the direct byte

MOV direct,#data Moves the immediate data to the direct byte

MOV @Ri,A Moves the accumulator to the indirect RAM

MOV @Ri,direct Moves the direct byte to the indirect RAM

MOV @Ri,#data Moves the immediate data to the indirect RAM

MOV DPTR,#data Moves a 16-bit data to the data pointer

MOVC A,@A+DPTR Moves the code byte relative to the DPTR to the accumulator (address=A+DPTR)

MOVC A,@A+PC Moves the code byte relative to the PC to the accumulator (address=A+PC)

MOVX A,@Ri Moves the external RAM (8-bit address) to the accumulator

MOVX A,@DPTR Moves the external RAM (16-bit address) to the accumulator

MOVX @Ri,A Moves the accumulator to the external RAM (8-bit address)

MOVX @DPTR,A Moves the accumulator to the external RAM (16-bit address)

PUSH direct Pushes the direct byte onto the stack

POP direct Pops the direct byte from the stack

XCH A,Rn Exchanges the register with the accumulator

XCH A,direct Exchanges the direct byte with the accumulator

XCH A,@Ri Exchanges the indirect RAM with the accumulator

XCHD A,@Ri Exchanges the low-order nibble indirect RAM with the accumulator

2.2.4 Logic Instructions

Logic instructions perform logic operations upon corresponding bits of two registers. After execution, the result is stored
in the first operand.

ANL A,Rn AND register to accumulator

ANL A,direct AND direct byte to accumulator

ANL A,@Ri AND indirect RAM to accumulator

ANL A,#data AND immediate data to accumulator

ANL direct,A AND accumulator to direct byte

ANL direct,#data AND immediae data to direct register

ORL A,Rn OR register to accumulator

ORL A,direct OR direct byte to accumulator

ORL A,@Ri OR indirect RAM to accumulator

ORL direct,A OR accumulator to direct byte

ORL direct,#data OR immediate data to direct byte

XRL A,Rn Exclusive OR register to accumulator

XRL A,direct Exclusive OR direct byte to accumulator

XRL A,@Ri Exclusive OR indirect RAM to accumulator

XRL A,#data Exclusive OR immediate data to accumulator

XRL direct,A Exclusive OR accumulator to direct byte

XORL direct,#data Exclusive OR immediate data to direct byte

CLR A Clears the accumulator

CPL A Complements the accumulator (1=0, 0=1)

SWAP A Swaps nibbles within the accumulator

RL A Rotates bits in the accumulator left

RLC A Rotates bits in the accumulator left through carry

RR A Rotates bits in the accumulator right

RRC A Rotates bits in the accumulator right through carry

2.2.5 Bit-oriented Instructions

Similar to logic instructions, bit-oriented instructions perform logic operations. The difference is that these are performed
upon single bits.

CLR C Clears the carry flag

CLR bit Clears the direct bit

SETB C Sets the carry flag

SETB bit Sets the direct bit

CPL C Complements the carry flag

CPL bit Complements the direct bit

ANL C,bit AND direct bit to the carry flag

ANL C,/bit AND complements of direct bit to the carry flag

ORL C,bit OR direct bit to the carry flag

ORL C,/bit OR complements of direct bit to the carry flag

MOV C,bit Moves the direct bit to the carry flag

MOV bit,C Moves the carry flag to the direct bit

Description of all 8051 instructions

Here is a list of the operands and their meanings:

A - accumulator;

Rn - is one of working registers (R0-R7) in the currently active RAM memory bank;

Direct - is any 8-bit address register of RAM. It can be any general-purpose register or a SFR (I/O port, control register etc.);

@Ri - is indirect internal or external RAM location addressed by register R0 or R1;

#data - is an 8-bit constant included in instruction (0-255);

#data16 - is a 16-bit constant included as bytes 2 and 3 in instruction (0-65535);

addr16 - is a 16-bit address. May be anywhere within 64KB of program memory;

addr11 - is an 11-bit address. May be within the same 2KB page of program memory as the first byte of the following instruction;

rel - is the address of a close memory location (from -128 to +127 relative to the first byte of the following instruction). On the basis of it, assembler computes the value to add or subtract from the number currently stored in the program counter;

bit - is any bit-addressable I/O pin, control or status bit; and

C - is carry flag of the status register (register PSW).

8051 Addressing Modes

8051 has four addressing modes.

1. Immediate Addressing :

Data is immediately available in the instruction.

For example -

ADD A, #77; Adds 77 (decimal) to A and stores in A

ADD A, #4DH; Adds 4D (hexadecimal) to A and stores in A

MOV DPTR, #1000H; Moves 1000 (hexadecimal) to data pointer

2. Bank Addressing or Register Addressing :

This way of addressing accesses the bytes in the current register bank. Data is available in the register specified in the instruction. The register bank is decided by 2 bits of Processor Status Word (PSW).

For example-

ADD A, R0; Adds content of R0 to A and stores in A

3.. Direct Addressing :
The address of the data is available in the
instruction. For example -

MOV A, 088H; Moves content of SFR TCON (address 088H)to A
4. Register Indirect Addressing :
The address of data is available in the R0 or R1 registers as specified in the
instruction. For example -

MOV A, @R0 moves content of address pointed by R0 to A
External Data Addressing :
Pointer used for external data addressing can be either R0/R1 (256 byte access) or
DPTR (64kbyte access).
For example -

MOVX A, @R0; Moves content of 8-bit address pointed by R0 to A

MOVX A, @DPTR; Moves content of 16-bit address pointed by DPTR to A
External Code Addressing :
Sometimes we may want to store non-volatile data into the ROM e.g. look-up tables.
Such data may require reading the code memory. This may be done as follows -

MOVC A, @A+DPTR; Moves content of address pointed by A+DPTR to A

MOVC A, @A+PC; Moves content of address pointed by A+PC to A

I/O Port Configuration
Each port of 8051 has bidirectional capability. Port 0 is called 'true bidirectional port' as it floats (tristated) when configured as input. Port-1, 2, 3 are called 'quasi bidirectional port'. Port-0 Pin Structure
Port -0 has 8 pins (P0.0-P0.7).
The structure of a Port-0 pin is shown in fig 4.10.

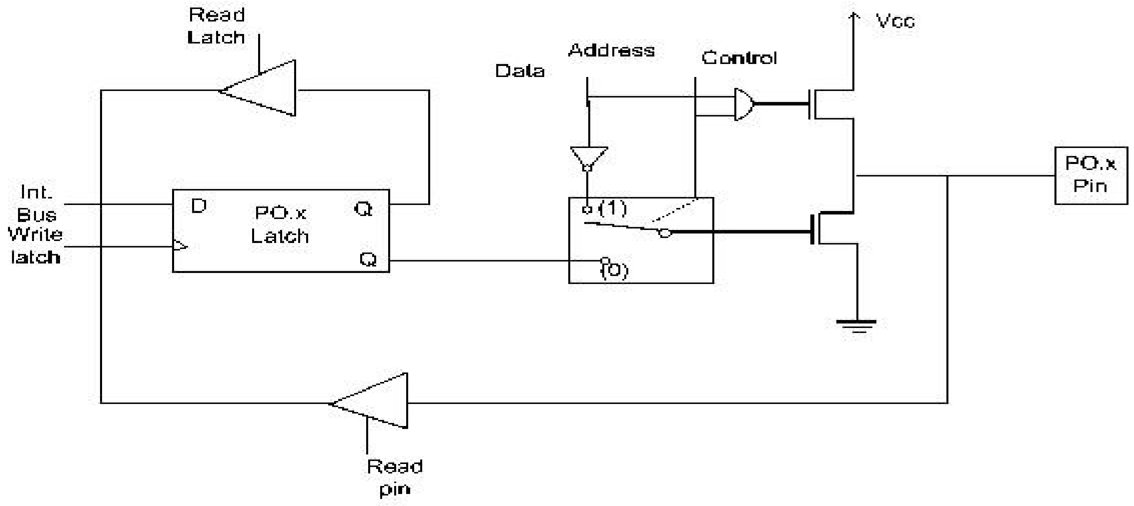


Fig 4.10: Port-0 Structure

Port-0 can be configured as a normal bidirectional I/O port or it can be used for address/data interfacing for accessing external memory. When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a normal bidirectional I/O port.

Let us assume that control is '0'. When the port is used as an input port, '1' is written to the latch. In this situation both the output MOSFETs are 'off'. Hence the output pin floats. This high impedance pin can be pulled up or low by an external source. When the port is used as an output port, a '1' written to the latch again turns 'off' both the output MOSFETs and causes the output pin to float. An external pull-up is required to output a '1'. But when '0' is written to the latch, the pin is pulled down by the lower MOSFET. Hence the output becomes zero.

When the control is '1', address/data bus controls the output driver MOSFETs. If the address/data bus (internal) is '0', the upper MOSFET is 'off' and the lower MOSFET is 'on'. The output becomes '0'. If the address/data bus is '1', the upper transistor is 'on' and the lower transistor is 'off'. Hence the output is '1'. Hence for normal address/data interfacing (for external memory access) no pull-up resistors are required.

Port-0 latch is written to with 1's when used for external memory
access. Port-1 Pin Structure
Port-1 has 8 pins (P1.1-P1.7) .The structure of a port-1 pin is shown in fig 4.11.

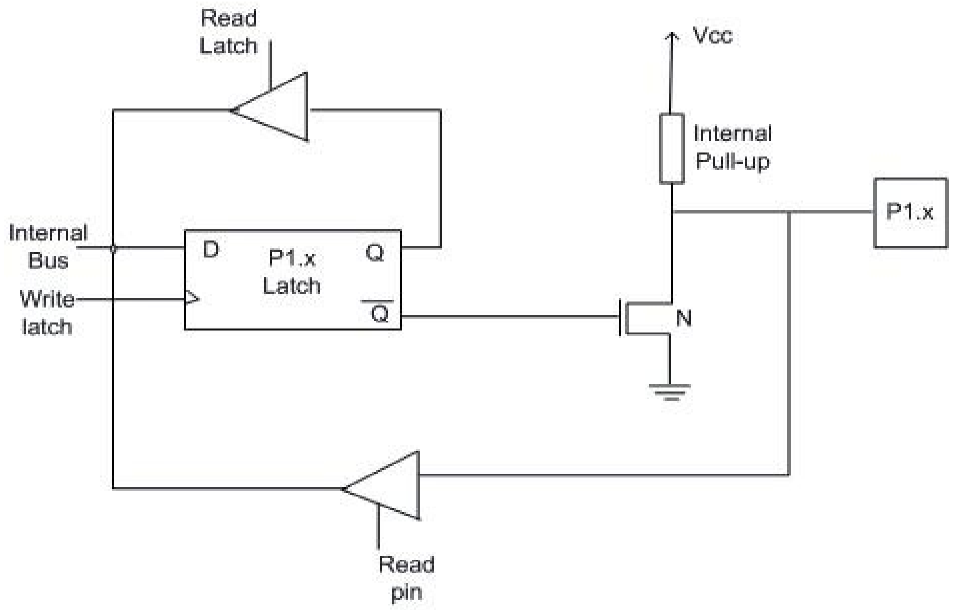


Fig 4.11 Port 1 Structure

Port-1 does not have any alternate function i.e. it is dedicated solely for I/O interfacing. When used as output port, the pin is pulled up or down through internal pull-up. To use port-1 as input port, '1' has to be written to the latch. In this input mode when '1' is written to the pin by the external device then it read fine. But when '0' is written to the pin by the external device then the external source must sink current due to internal pull-up. If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.

PORT 2 Pin Structure

Port-2 has 8-pins (P2.0-P2.7) . The structure of a port-2 pin is shown in fig 4.12.

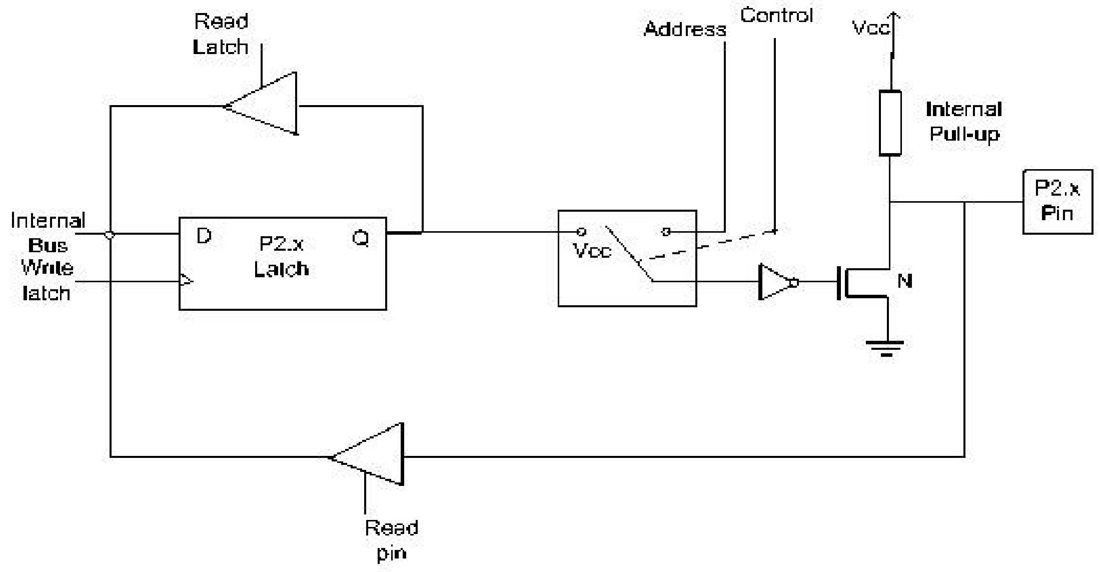


Fig 4.12 Port 2 Structure

Port-2 is used for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1. Port-2 latch remains stable when Port-2 pin are used for external memory access. Here again due to internal pull-up there is limited current driving capability.

PORT 3 Pin Structure

Port-3 has 8 pin (P3.0-P3.7) . Port-3 pins have alternate functions. The structure of a port-3 pin is shown in fig 4.13.

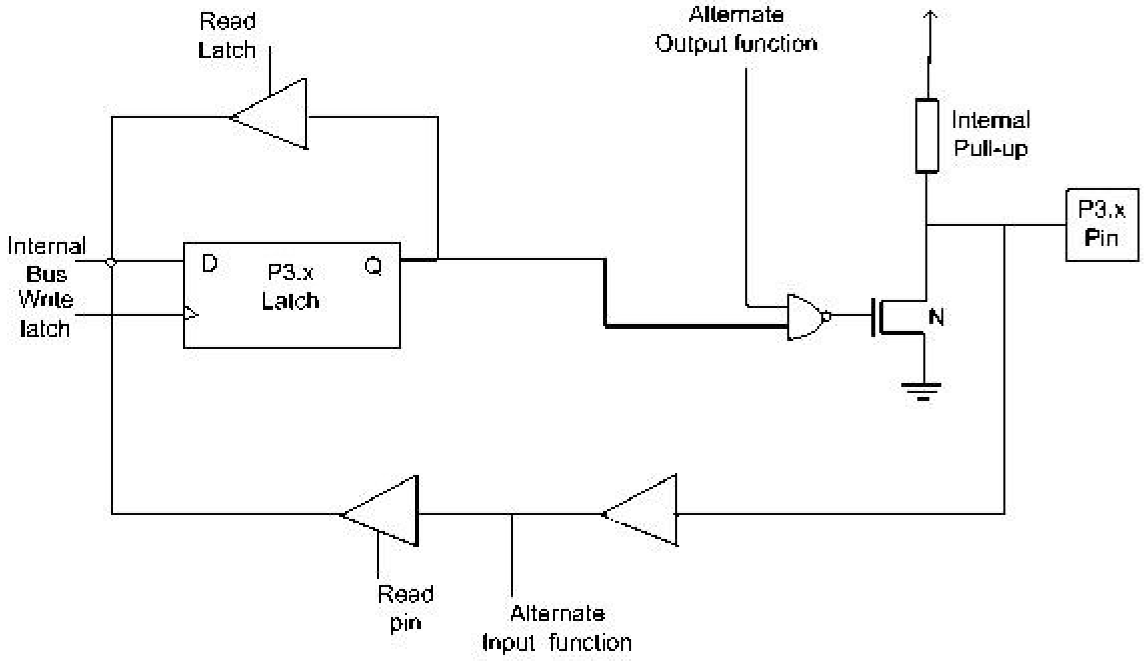


Fig 4.13 Port 3 Structure

Each pin of Port-3 can be individually programmed for I/O operation or for alternate function. The alternate function can be activated only if the corresponding latch has been

written to '1'. To use the port as input port, '1' should be written to the latch. This port also has internal pull-up and limited current driving capability.
Alternate functions of Port-3 pins are -

P3.0	RxD
P3.1	TxD
P3.2	$\overline{INT0}$
P3.3	$\overline{INT1}$
P3.4	T0
P3.5	T1
P3.6	$\overline{WR}$
P3.7	$\overline{RD}$

Note:
1) Port 1, 2, 3 each can drive 4 LS TTL inputs.
2) Port-0 can drive 8 LS TTL inputs in address /data mode. For digital output port, it needs external pull-up resistors.
3) Ports-1,2and 3 pins can also be driven by open-collector or open-drain outputs.
4) Each Port 3 bit can be configured either as a normal I/O or as a special function bit.

8051 MICROCONTROLLER PROGRAMS

1.8 BIT ADDITION USING INTERNAL MEMORY

Memory Address	Hex code	Label	Mnemonics		Comments
			Opcode	Operand	
8000	E5,40		MOV	A,40	Move the content of 40 to accumulator
8002	A8,41		MOV	R0,41	Move the content of 41 to ‘R0’ register
8004	28		ADD	A,R0	Add the content of ‘R0’ and ‘A’
8005	F5,42		MOV	42,A	Move the content of accumulator to 42
8007	74,00		MOV	A,#00	Initialize the accumulator
8009	34,00		ADDC	A,#00	Add the content of A and 00 with carry
800B	F5,43		MOV	43,A	Move the content of accumulator to 43
800D	12,00,BB		LCALL	00BB	Halt the program

2. 8 BIT ADDITION USING EXTERNAL MEMORY

Memory Address	Hex code	Label	Mnemonics		Comments
			Opcode	Operand	
8000	90,91,00		MOV	DPTR,#9100	Initialize the data pointer
8003	E0		MOVX	A,@DPTR	Move the content of DPTR to Acc.

8004	F8		MOV	R0,A	Move the content of A to R0
8005	A3		INC	DPTR	Increment the data pointer
8006	E0		MOVX	A,@DPTR	Move the content of DPTR to Acc.
8007	28		ADD	A,R0	Add the content of ‘R0’ and ‘A’
8008	A3		INC	DPTR	Increment the data pointer
8009	F0		MOVX	@DPTR,A	Move the content of A to DPTR
800A	74,00		MOV	A,#00	Initialize the accumulator
800C	34,00		ADDC	A,#00	Add the content of A and 00 with carry
800E	A3		INC	DPTR	Increment the data pointer
800F	F0		MOVX	@DPTR,A	Move the content of A to DPTR
8010	12,00,BB		LCALL	00BB	Halt the program

3. 8 BIT SUBTRACTION USING INTERNAL MEMORY

Memory Address	Hex code	Label	Mnemonics		Comments
			Opcode	Operand	
8000	C3		CLR	C	Clear the Carry flag
8001	E5,40		MOV	A,40	Move the content of 40 to accumulator
8003	A8,41		MOV	R0,41	Move the content of 41 to ‘R0’ register
8005	98		SUBB	A,R0	Subtract the content of ‘R0’ from ‘A’
8006	F5,42		MOV	42,A	Move the content of accumulator to 42
8008	12,00,BB		LCALL	00BB	Halt the program

4. 8 BIT SUBTRACTION USING EXTERNAL MEMORY

Memory Address	Hex code	Label	Mnemonics		Comments
			Opcode	Operand	
8000	C3		CLR	C	Clear the Carry flag
8001	90,91,00		MOV	DPTR,#9100	Initialize the data pointer
8004	E0		MOVX	A,@DPTR	Move the content of DPTR to Acc.
8005	F8		MOV	R0,A	Move the content of A to R0
8006	A3		INC	DPTR	Increment the data pointer
8007	E0		MOVX	A,@DPTR	Move the content of DPTR to Acc.
8008	98		SUBB	A,R0	Subtract the content of ‘R0’ from ‘A’
8009	A3		INC	DPTR	Increment the data pointer
800A	F0		MOVX	@DPTR,A	Move the content of A to DPTR
801B	12,00,BB		LCALL	00BB	Halt the program

5. 8 BIT MULTIPLICATION USING INTERNAL MEMORY

Memory Address	Hex code	Label	Mnemonics		Comments
			Opcode	Operand	
8000	E5,40		MOV	A,40	Move the content of 40 to accumulator
8002	85,41,F0		MOV	0F0,41	Move the content of 41 to ‘B’ register

8005	A4		MUL	AB	Multiply the content of ‘A’ and ‘B’
8006	F5,42		MOV	42,A	Move the content of accumulator to 42
8008	E5,F0		MOV	A,0F0	Move the content of ‘B’ to accumulator
800A	F5,43		MOV	43,A	Move the content of accumulator to 43
800C	12,00,BB		LCALL	00BB	Halt the program

6. 8 BIT MULTIPLICATION USING EXTERNAL MEMORY

Memory Address	Hex code	Label	Mnemonics		Comments
			Opcode	Operand	
8000	90,91,00		MOV	DPTR,#9100	Initialize the data pointer
8003	E0		MOVX	A,@DPTR	Move the content of DPTR to Acc.
8004	F5,F0		MOV	0F0,A	Move the content of ‘A’ to ‘B’ register
8006	A3		INC	DPTR	Increment the data pointer
8007	E0		MOVX	A,@DPTR	Move the content of DPTR to Acc.
8008	A4		MUL	AB	Multiply the content of ‘A’ and ‘B’
8009	A3		INC	DPTR	Increment the data pointer
800A	F0		MOVX	@DPTR,A	Move the content of ‘A’ to DPTR
800B	E5,F0		MOV	A,0F0	Move the content of ‘B’ to accumulator
800D	A3		INC	DPTR	Increment the data pointer
800E	F0		MOVX	@DPTR,A	Move the content of A to DPTR
800F	12,00,BB		LCALL	00BB	Halt the program

7. 8 BIT DIVISION USING INTERNAL MEMORY

Memory Address	Hex code	Label	Mnemonics		Comments
			Opcode	Operand	
8000	E5,40		MOV	A,40	Move the content of 40 to accumulator
8002	85,41,F0		MOV	0F0,41	Move the content of 41 to ‘B’ register
8005	84		DIV	AB	Divide the content of ‘A’ and ‘B’
8006	F5,42		MOV	42,A	Move the content of accumulator to 42
8008	E5,F0		MOV	A,0F0	Move the content of ‘B’ to accumulator
800A	F5,43		MOV	43,A	Move the content of accumulator to 43
800C	12,00,BB		LCALL	00BB	Halt the program

UNIT 5 APPLICATIONS BASED ON 8085 AND 8051 9 Hrs.
Interfacing Basic concepts, interfacing LED, 7 segment LED, Stepper motor control system, Temperature control system, Traffic light control system, Motor speed control system, Waveform generation, Interfacing LCD.

Traffic light control system using 8085

The traffic lights placed at the road crossings can be automatically switched ON/OFF in the desired sequence using the microprocessor system. The system can also have a manual control option, so that during heavy traffic (or during4raffic jam) the duration of ON/OFF time can be varied by the operator.

A typical traffic light control system (demonstration type) is shown in fig 1. The systems have been developed using 8085 as CPU. The system has EPROM memory for system program storage and RAM memory for stack operation. For manual control a keyboard have been provided. It will be helpful for the operator if the direction 6ftraffic flow is displayed during manual control. Hence 7-segment LEDs are interfaced to display the direction of traffic flow both during manual and automatic mode.

The primary function of the microprocessor in the system is to switch ON/OFF the Red/Yellow/Green lights is the specified sequence. In the demonstration system of fig, Red/Yellow/Green LEDs are provided instead of lights (lamps). The LEDs are interfaced to the system through buffer (74LS245) and ports of 8255.

In the practical implementation scheme the lights can be turned ON/OFF using driver transistors and relays. In practical implementation the output of buffer (74LS245) can be connected to the driver transistor. A reverse biased diode is connected across relay coil to prevent relay chattering (for free-wheeling action).

The microprocessor send High through a port line to switch ON the light and LOW to switch OFF the light. A switching schedule (or sequence) can be developed as shown in Table. In this switching sequence it is assumed that the traffic is allowed only in one direction at time. In table, "1" represents ON condition and "0" represents OFF condition. These 1's and O's
Can be directly output to 8255 ports to switch ON/OFF the light. A flowchart for traffic light control program is shown in fig 3.

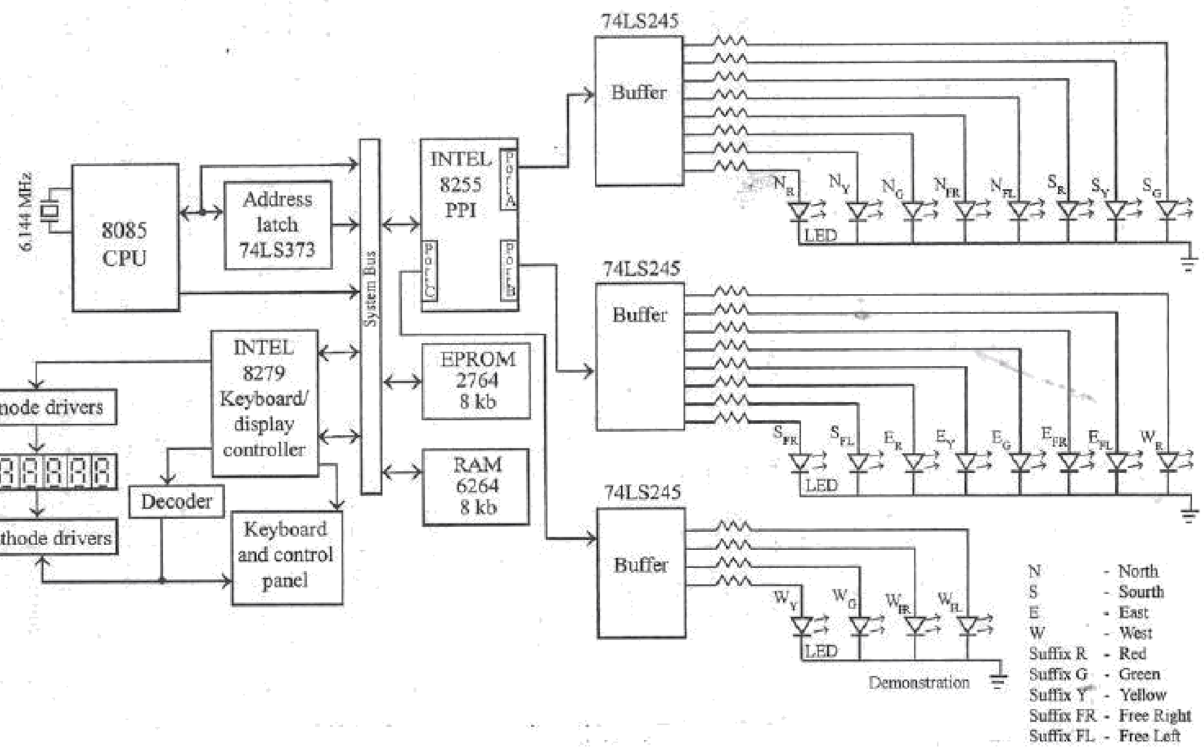
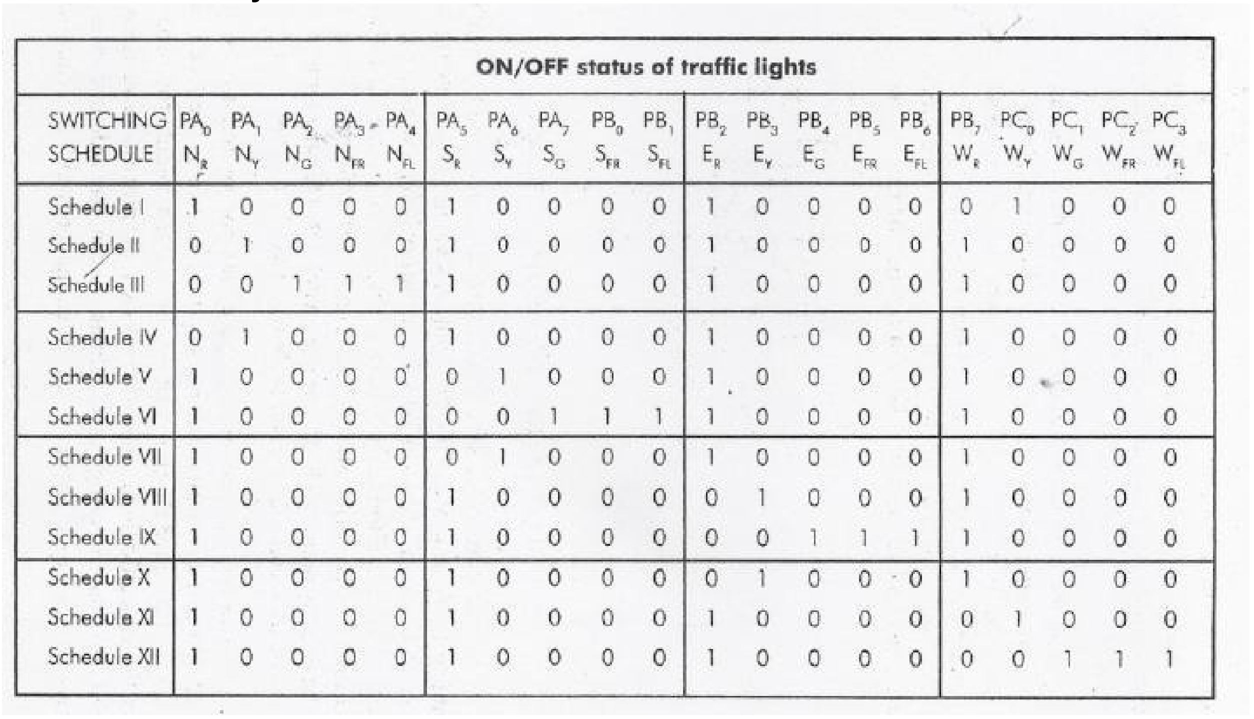


Fig.1: 8085 Microprocessor based Traffic light control system.



ON/OFF status of traffic lights

SWITCHING SCHEDULE	PA_0 N_R	PA_1 N_Y	PA_2 N_G	PA_3 N_{FR}	PA_4 N_{FL}	PA_5 S_R	PA_6 S_Y	PA_7 S_G	PB_0 S_{FR}	PB_1 S_{FL}	PB_2 E_R	PB_3 E_Y	PB_4 E_G	PB_5 E_{FR}	PB_6 E_{FL}	PB_7 W_R	PC_0 W_Y	PC_1 W_G	PC_2 W_{FR}	PC_3 W_{FL}
Schedule I	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0
Schedule II	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0
Schedule III	0	0	1	1	1	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0
Schedule IV	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0
Schedule V	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0
Schedule VI	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0	0
Schedule VII	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0
Schedule VIII	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0
Schedule IX	1	0	0	0	0	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0
Schedule X	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0
Schedule XI	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0
Schedule XII	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	1	1

Fig. 2: Switching schedule for Traffic light

The processor can output the codes for switching the lights for scheduleI and then waits. After a specified time delay the processor output the codes for schedule-II and so on. For each schedule the processor can wait for a specified time. After schedule-XII, the processor can again return to schedule-I. On observing the schedules we can conclude that three different delay routines are sufficient for implementing the twelve switching schedules.

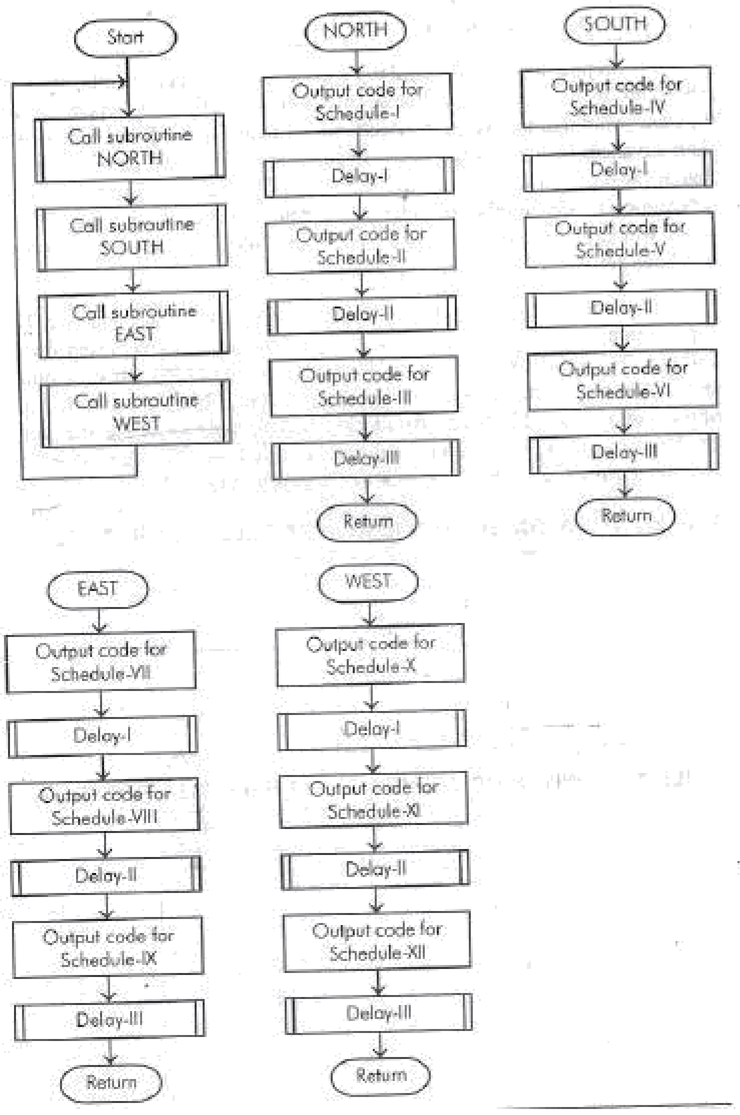

Start
Call subroutine NORTH
Call subroutine SOUTH
Call subroutine EAST
Call subroutine WEST
NORTH
Output code for Schedule-I
Delay-I
Output code for Schedule-II
Delay-II
Output code for Schedule-III
Delay-III
Return
SOUTH
Output code for Schedule-IV
Delay-I
Output code for Schedule-V
Delay-II
Output code for Schedule-VI
Delay-III
Return
EAST
Output code for Schedule-VII
Delay-I
Output code for Schedule-VIII
Delay-II
Output code for Schedule-IX
Delay-III
Return
WEST
Output code for Schedule-X
Delay-I
Output code for Schedule-XI
Delay-II
Output code for Schedule-XII
Delay-III
Return

Fig.3: Flow chart for Traffic light control system.

Temperature controller using 8085

The microprocessor based temperature control system can be used for automatic control of the temperature of a body. A simplified block diagram of 8085 microprocessor based temperature control system is shown in fig 4 . The system consist of 8085 microprocessor as CPU, ERROM&RAM memory for program & data storage, INTEL 8279 for keyboard and display Interface, ADC, DAC, INTEL 8255 for I/O' ports, Amplifiers, Signal conditioning circuit, Temperature sensor and Supply control circuit. In this system the temperature is controlled by controlling the power input to the heating element.

The temperature of the body is measured using a temperature sensor. The different types of temperature sensor that can be used for temperature measurement are Thermo-couple, Thermistors , PN-junctions, IC sensors. This sensor will convert the input temperature to Proportional analog voltage or current. The output signal of the sensor will be a weak signal and so has to be amplified by using high input impedance op-amp. Then the analog signal is scaled to suitable level by the signal conditioning circuit.

The microprocessor can process only digital signals and so the analog signal from signal conditioning circuit cannot be read by the processor directly. The system has an analog-to-digital converter (ADC) to convert the analog signal to proportional digital data. In this system the ADC is interfaced to 8085 processor through port-A of 8255. The 8085 processor send signal to ADC to start conversion and at the end of conversion it read the digital data from the port-A of 8255.

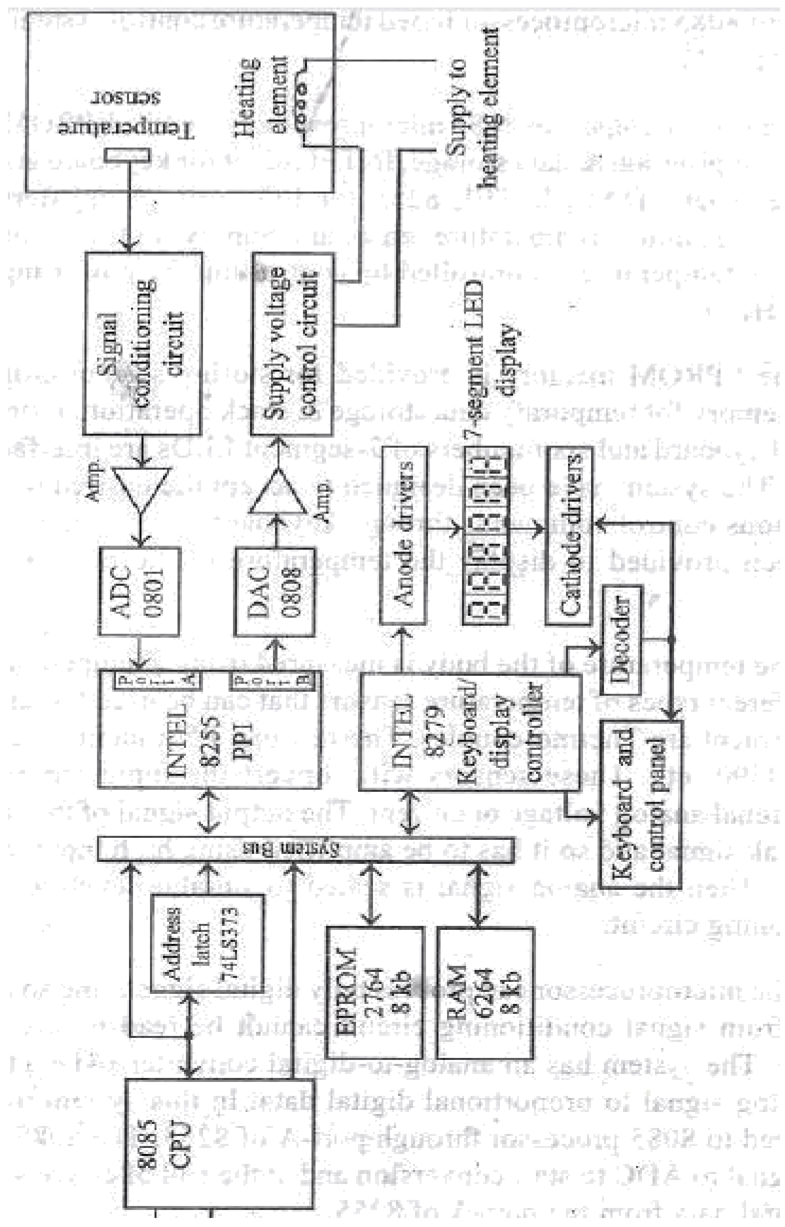


Fig.4 : 8085 Microprocessor based Temperature control system.

The 8085 processor calculate the actual temperature using the input data and display it on the 7-segment LED. Also, the processor compare the desired temperature with actual temperature (The operator can enter the desired temperature through keyboard) and calculate the error (the difference between actual temperature and desired temperature).

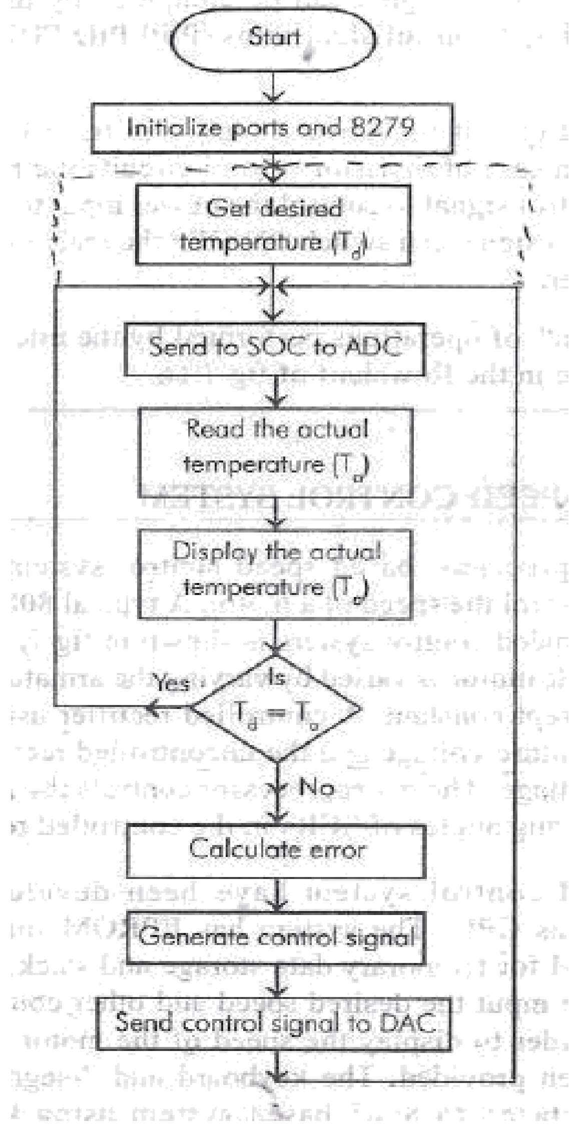

Start
Initialize ports and 8279
Get desired temperature (T_d)
Send to SOC to ADC
Read the actual temperature (T_a)
Display the actual temperature (T_a)
Is $T_d = T_a$
Yes
No
Calculate error
Generate control signal
Send control signal to DAC

Fig.5: Flow chart for Temperature control system.

The error is used to compute a digital control signal, which is converted to analog control signal by DAC. The DAC is interfaced to the system through port-B of 8255. The analog control signal produced by DAC is used to control the power supply of the heating element of the body.

The digital control signal can be computed by the 8085 processor using different digital control algorithms (PIPI/PID/FUZZY logic control algorithms).

The control circuit for power supply can be either thyristor based circuit or relay. In case of thyristor control circuits the firing angle can be varied by the control signal to control the power input to the heater.

Stepper motor controller system using 8085

The stepper motors are popularly used in computer peripherals, plotters, robots and machine tools for 'precise incremental rotation. In stepper motor, the stator windings are excited by electrical pulses and for each pulse the motor shaft advances by one angular step. (Single the stepper motor can be driven by digital pulses; it is also called digital motor). The step size in the motor is determined by the number of poles in the rotor and the number 'of pairs of stator windings (one pair of stator winding is called one 'phase). The stator windings are also called control windings.

The motor is controlled by switching ON/OFF the control winding. The popular stepper motor used for demonstration in laboratories has a step size of 1.8° (i.e, 200 steps per revolution). This motor consist of four stator winding and require four switching sequence as shown in table 1. The basic step size of the motor is called full-step. By altering the switching sequence, the motor can be made to run with incremental motion of half the full-step value.

A typical stepper motor control system is shown in fig 6. a two-phase or four winding stepper motor is show in fig 6. The system consists of 8085 microprocessor as CPU, EPROM and RAM memory for program &data storage and for stack. Using INTEL 8279, a keyboard and six number of 7-segment LED display has been interfaced in the system. Through the keyboard the operator can issue commands to control the system. The LED display has been provided to display messages to the operator. The windings of stepper motor connected to the collector of Darlington pair transistors. The transistors are switched ON/OFF by the microprocessor through the ports of 8255 and buffer (74LS245). A free wheeling diode is connected across each winding for fast switching. The flowchart for the operational flow of the stepper motor control system is shown in fig 7. The processor has to output a switching sequence and wait for I to 5 msec before Sending next switching sequence. (The delay is necessary to allow be motor transients to die-out).

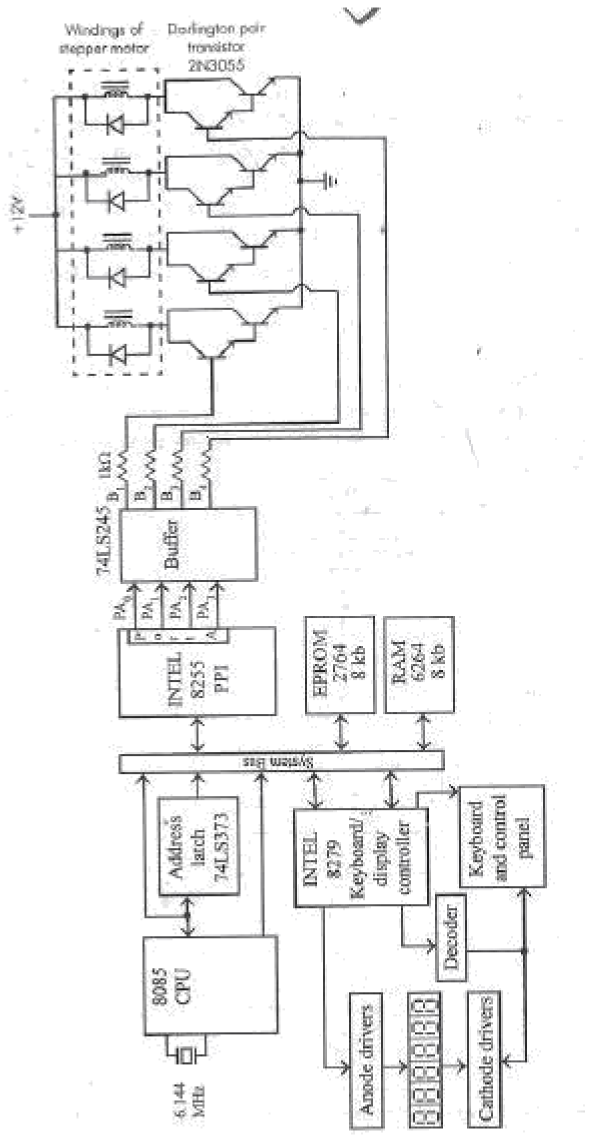
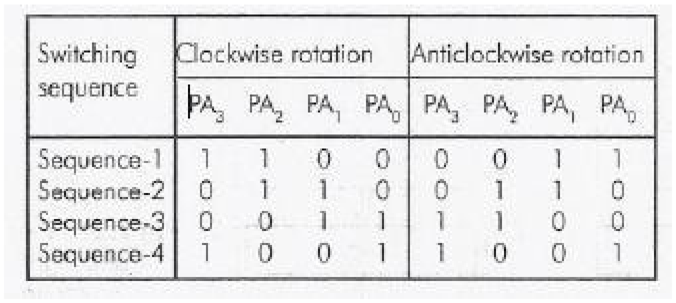


Fig.6 : 8085 Microprocessor based StepperMotor control system.

TABLE 1: Switching sequence for Full Step Rotation



Switching sequence	Clockwise rotation				Anticlockwise rotation			
	PA_3	PA_2	PA_1	PA_0	PA_3	PA_2	PA_1	PA_0
Sequence-1	1	1	0	0	0	0	1	1
Sequence-2	0	1	1	0	0	1	1	0
Sequence-3	0	0	1	1	1	1	0	0
Sequence-4	1	0	0	1	1	0	0	1

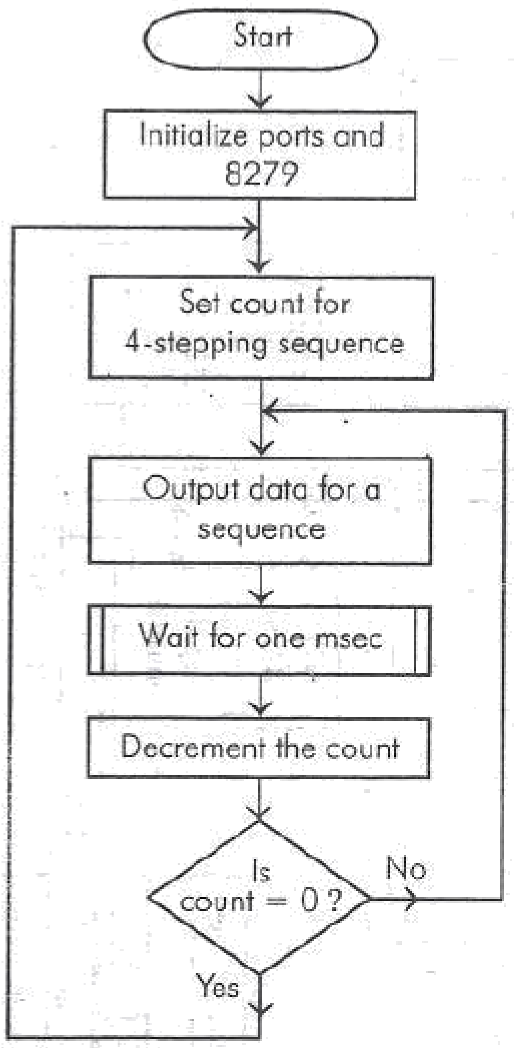
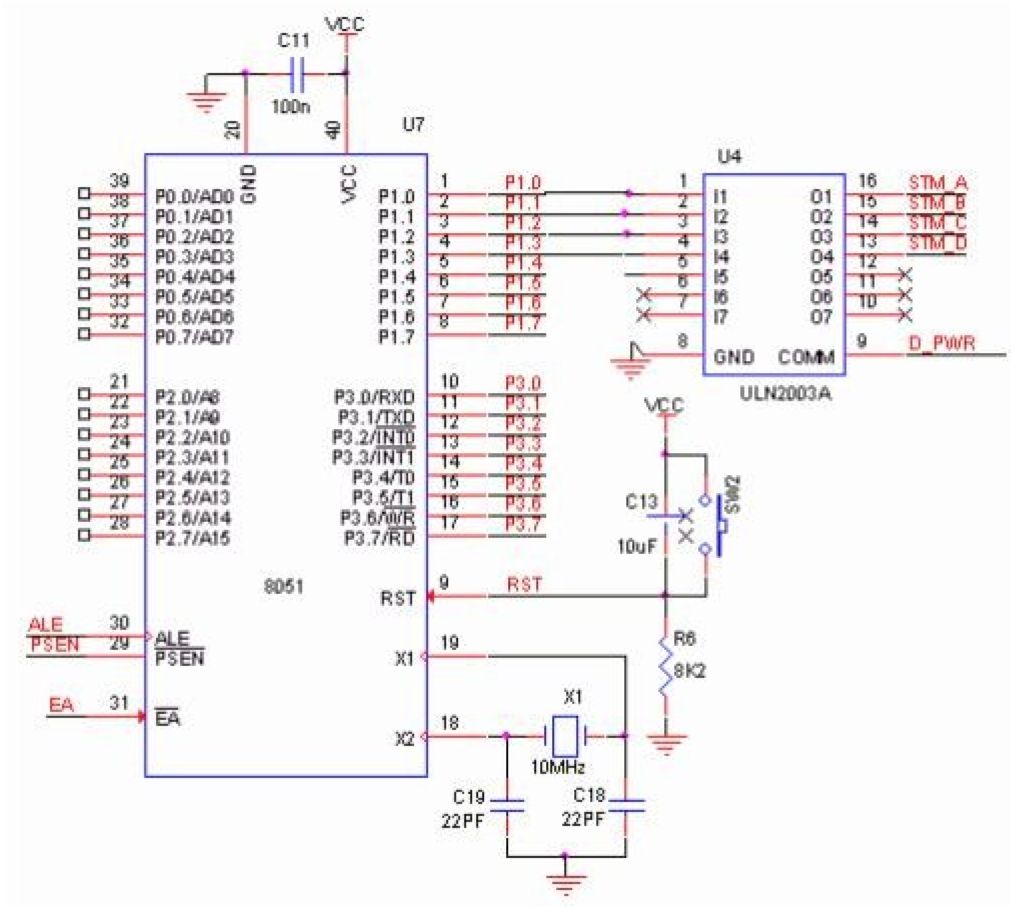

Start
Initialize ports and 8279
Set count for 4-stepping sequence
Output data for a sequence
Wait for one msec
Decrement the count
Is count = 0 ?
No
Yes

Fig.7: Flow chart for Stepper motor control system.

INTERFACING STEPPER MOTOR with 8051

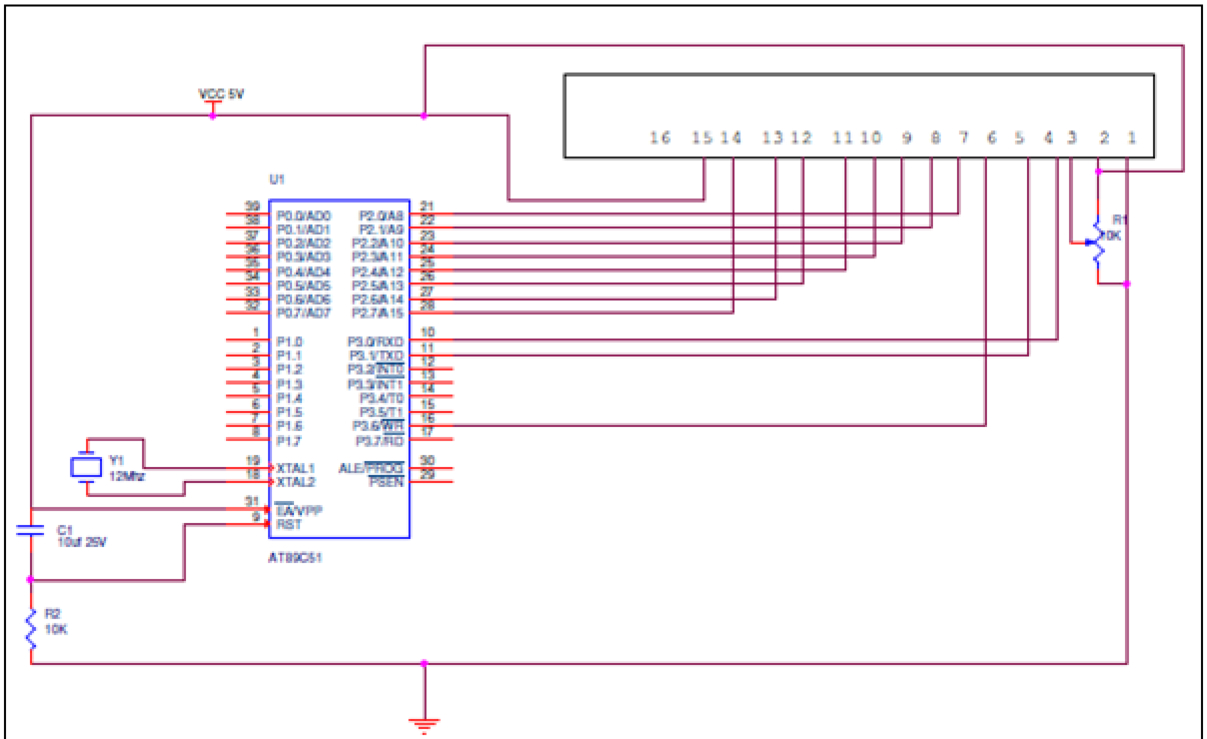
The Stepper Motor to microcontroller. As you can see the stepper motor is connected with Microcontroller output port pins through a ULN2803A array. So when the microcontroller is giving pulses with particular frequency to ls293A, the motor is rotated in clockwise or anticlockwise. Fig. 1 Interfacing Stepper Motor to Microcontroller

We now want to control a stepper motor in 8051 trainer kit. It works by turning ON & OFF a four I/O port lines generating at a particular frequency. The 8051 trainer kit has three numbers of I/O port connectors, connected with I/O Port lines (P1.0 – P1.7),(p3.0 – p3.7) to rotate the stepper motor. Ls293d is used as a driver for port I/O lines, drivers output connected to stepper motor, connector provided for external power supply if needed.



Interfacing 16×2 LCD with 8051

We use LCD display for the messages for more interactive way to operate the system or displaying error messages etc. interfacing LCD to microcontroller is very easy if you understanding the working of LCD, in this session I will not only give the information of LCD and also provide the code in C language which is working fine without any errors.



Circuit Diagram of Interfacing LCD to AT89C51

LCD: 16×2 Liquid Crystal Display which will display the 32 characters at a time in two rows (16 characters in one row). Each character in the display of size 5×7 pixel matrix, Although this matrix differs for different 16×2 LCD modules if you take JHD162A this matrix goes to 5×8. This matrix will not be same for all the 16×2 LCD modules. There are 16 pins in the LCD module, the pin configuration us given below

Follow these simple steps for displaying a character or data

- E=1; enable pin should be high
- RS=1; Register select should be high
- R/W=0; Read/Write pin should be low.

To send a command to the LCD just follows these steps:

- E=1; enable pin should be high
- RS=0; Register select should be low
- R/W=1; Read/Write pin should be high.

The crystal oscillator is connected to XTAL1 and XTAL2 which will provide the system clock to the microcontroller the data pins and remaining pins are connected to the microcontroller as shown in the circuit. The potentiometer is used to adjust the contrast of the LCD. You can connect data pins to any port. If you are connecting to port0 then you have to use pull up registers. The enable, R/W and RS pins are should be connected to the 10, 11 and 16 (P3.3, P3.4 and P3.5).