UNIT – I

Evolution and Architecture of Microprocessors 8085 and 8086

1-1.Evolution of Microprocessor 8085

(i) 4-bit Microprocessors: The first microprocessor was introduced in 1971 by Intel Corp. It was named Intel 4004 as it was a 4 bit processor. It was a processor on a single chip. It could perform simple arithmetic and logic operations such as addition, subtraction, boolean AND and boolean OR. It had a control unit capable of performing control functions like fetching an instruction from memory, decoding it, and generating control pulses to execute it. It was able to operate on 4 bits of data at a time.This first microprocessor was quite a success in industry. Soon other microprocessors were also introduced. Intel introduced the enhanced version of 4004, the 4040. Some other 4 bit processors are International's PPS4 and Thoshiba's T3472.

(ii) 8-bit Microprocessors: The first 8 bit microprocessor which could perform arithmetic and logic operations on 8 bit words was introduced in 1973 again by Intel. This was Intel 8008 and was later followed by an improved version, Intel 8088. Some other 8 bit processors are Zilog-80 and Motorola M6800.

(iii) 16-bit Microprocessors: The 8-bit processors were followed by 16 bit processors. They are Intel 8086 and 80286.

(iv) 32-bit Microprocessors: The 32 bit microprocessors were introduced by several companies but the most popular one is Intel 80386.

(v) Pentium Series: Instead of 80586, Intel came out with a new processor namely Pentium processor. Its performance is closer to RISC performance. Pentium was followed by Pentium Pro CPU. Pentium Pro allows allow multiple CPUs in a single system in order to achive multiprocessing. The MMX extension was added to Pentium Pro and the result was Pentiuum II. The low cost version of Pentium II is Celeron. The Pentium III provided high performance floating point operations for certain types of computations by using the SIMD extensions to the instruction set. These new instructions makes the Pentium III faster than high-end RISC CPUs. Interestingly Pentium IV could not execute code faster than the Pentium III when running at the same clock frequency. So Pentium IV had to speed up by executing at a much higher clock frequency.

1.2. Computer & Classification of Computers:

Computer is an electronic device which has many units like Input unit, Control unit and Output unit. Input unit consists of input devices like keyboard, mouse, scanner, light pen, etc., Output unit consists of output devices like printer, monitor, etc., Control unit controls all the actions of computer which consists of memory unit, Arithmetic and logic unit. A computer is one of the most brilliant inventions of mankind. Depending on the processing power and size of computers, they have been classified under various types.

(a) Classification of Computers on the basis of operational principle

Based on the operational principle of computers, they are categorized as analog, digital and hybrid computers.

(i). Analog Computers: These are almost extinct today. These are different from a digital computer because an analog computer can perform several mathematical operations simultaneously. It uses continuous variables for mathematical operations and utilizes mechanical or electrical energy.

(ii). Digital Computers: They use digital circuits and are designed to operate on two states, namely bits 0 and 1. They are analogous to states ON and OFF. Data on these computers is represented as a series of 0s and 1s. Digital computers are suitable for complex computation and have higher processing speeds. They are programmable. Digital computers are either general purpose computers or special purpose ones. General purpose computers, as their name suggests, are designed for specific types of data processing while general purpose computers are meant for general use.

(iii). Hybrid Computers: These computers are a combination of both digital and analog computers. In this type of computers, the digital segments perform process control by conversion of analog signals to digital ones.

(b) Classification on the basis of types:

(i). Mainframe Computers: Large organizations use mainframes for highly critical applications such as bulk data processing and ERP. Most of the mainframe computers have capacities to host multiple operating systems and operate as a number of virtual machines. They can substitute for several small servers.

(ii). Microcomputers: A computer with a microprocessor and its central processing unit is known as a microcomputer. They do not occupy space as much as mainframes do. When supplemented with a keyboard and a mouse, microcomputers can be called personal computers. A monitor, a keyboard and other similar input-output devices, computer memory in the form of RAM and a power supply unit come packaged in a microcomputer. These computers can fit on desks or tables and prove to be the best choice for single-user tasks.

(iii). Personal Computers: Personal computers come in different forms such as desktops, laptops and personal digital assistants. Let us look at each of these types of computers.

(iv) Desktops: A desktop is intended to be used on a single location. The spare parts of a desktop computer are readily available at relatively lower costs. Power consumption is not as critical as that in laptops. Desktops are widely popular for daily use in the workplace and households.

(v)Laptops: Similar in operation to desktops, laptop computers are miniaturized and optimized for mobile use. Laptops run on a single battery or an external adapter that charges the computer batteries. They are enabled with an inbuilt keyboard, touch pad acting as a mouse and a liquid crystal display. Their portability and capacity to operate on battery power have proven to be of great help to mobile users.

(vi) Notebooks: They fall in the category of laptops, but are inexpensive and relatively smaller in size. They had a smaller feature set and lesser capacities in comparison to regular laptops,

(v) Personal Digital Assistants (PDAs): It is a handheld computer and popularly known as a palmtop. It has a touch screen and a memory card for storage of data. PDAs can also be used as portable audio players, web browsers and smart phones. Most of them can access the Internet by means of Bluetooth or Wi-Fi communication.

(vi) Minicomputers: In terms of size and processing capacity, minicomputers lie in between mainframes and microcomputers. Minicomputers are also called mid-range systems or workstations. The term began to be popularly used in the 1960s to refer to relatively smaller third generation computers. They took up the space that would be needed for a refrigerator or two and used transistor and core memory technologies. The 12-bit PDP-8 minicomputer of the Digital Equipment Corporation was the first successful minicomputer.

(vii) Servers: They are computers designed to provide services to client machines in a computer network. They have larger storage capacities and powerful processors. Running on them are programs that serve client requests and allocate resources like memory and time to client machines. Usually they are very large in size, as they have large processors and many hard drives. They are designed to be fail-safe and resistant to crash.

(viii) Supercomputers: The highly calculation-intensive tasks can be effectively performed by means of supercomputers. Quantum physics, mechanics, weather forecasting, molecular theory are best studied by means of supercomputers. Their ability of parallel processing and their well-designed memory hierarchy give the supercomputers, large transaction processing powers.

(ix) Wearable Computers: A record-setting step in the evolution of computers was the creation of wearable computers. These computers can be worn on the body and are often used in the study of behavior modeling and human health. Military and health professionals have incorporated wearable computers into their daily routine, as a part of such studies. When the users' hands and sensory organs are engaged in other activities, wearable computers are of great help in tracking human actions. Wearable computers do not have to be turned on and off and remain in operation without user intervention.

(x) Tablet Computers: Tablets are mobile computers that are very handy to use. They use the touch screen technology. Tablets come with an onscreen keyboard or use a stylus or a digital pen. Apple's iPad redefined the class of tablet computers.

1.3. Pin diagram and Pin description of 8085

8085 is a 40 pin IC, DIP package. The signals from the pins can be grouped as follows

(i) Power supply and clock signals
(ii) Address bus
(iii) Data bus
(iv) Control and status signals
(v) Interrupts and externally initiated signals
(vi) Serial I/O ports

(i) Power supply and Clock frequency signals:

- Vcc + 5 volt power supply
- Vss Ground
- X1, X2 : Crystal or R/C network or LC network connections to set the frequency of internal clock generator.
- The frequency is internally divided by two. Since the basic operating timing frequency is 3 MHz, a 6 MHz crystal is connected externally.
- CLK (output)-Clock Output is used as the system clock for peripheral and devices interfaced with the microprocessor.

(ii) Address Bus:

- A8 - A15 (output; 3-state). It carries the most significant 8 bits of the memory address or the 8 bits of the I/O address;

(iii). Multiplexed Address / Data Bus:

- AD0 - AD7 (input/output; 3-state). These multiplexed set of lines used to carry the lower order 8 bit address as well as data bus. During the opcode fetch operation, in the first clock cycle, the lines deliver the lower order address A0 - A7.
- In the subsequent IO / memory, read / write clock cycle the lines are used as data bus.
- The CPU may read or write out data through these lines.

(iv). Control and Status signals:

- ALE (output) - Address Latch Enable. This signal helps to capture the lower order address presented on the multiplexed address / data bus.
- RD (output 3-state, active low) - Read memory or IO device. This indicates that the selected memory location or I/O device is to be read and that the data bus is ready for accepting data from the memory or I/O device.

- WR (output 3-state, active low) - Write memory or IO device. This indicates that the data on the data bus is to be written into the selected memory location or I/O device.
- IO/M (output) - Select memory or an IO device. This status signal indicates that the read / write operation relates to whether the memory or I/O device. It goes high to indicate an I/O operation.It goes low for memory operations.

(v). Status Signals:

- It is used to know the type of current operation of the microprocessor.

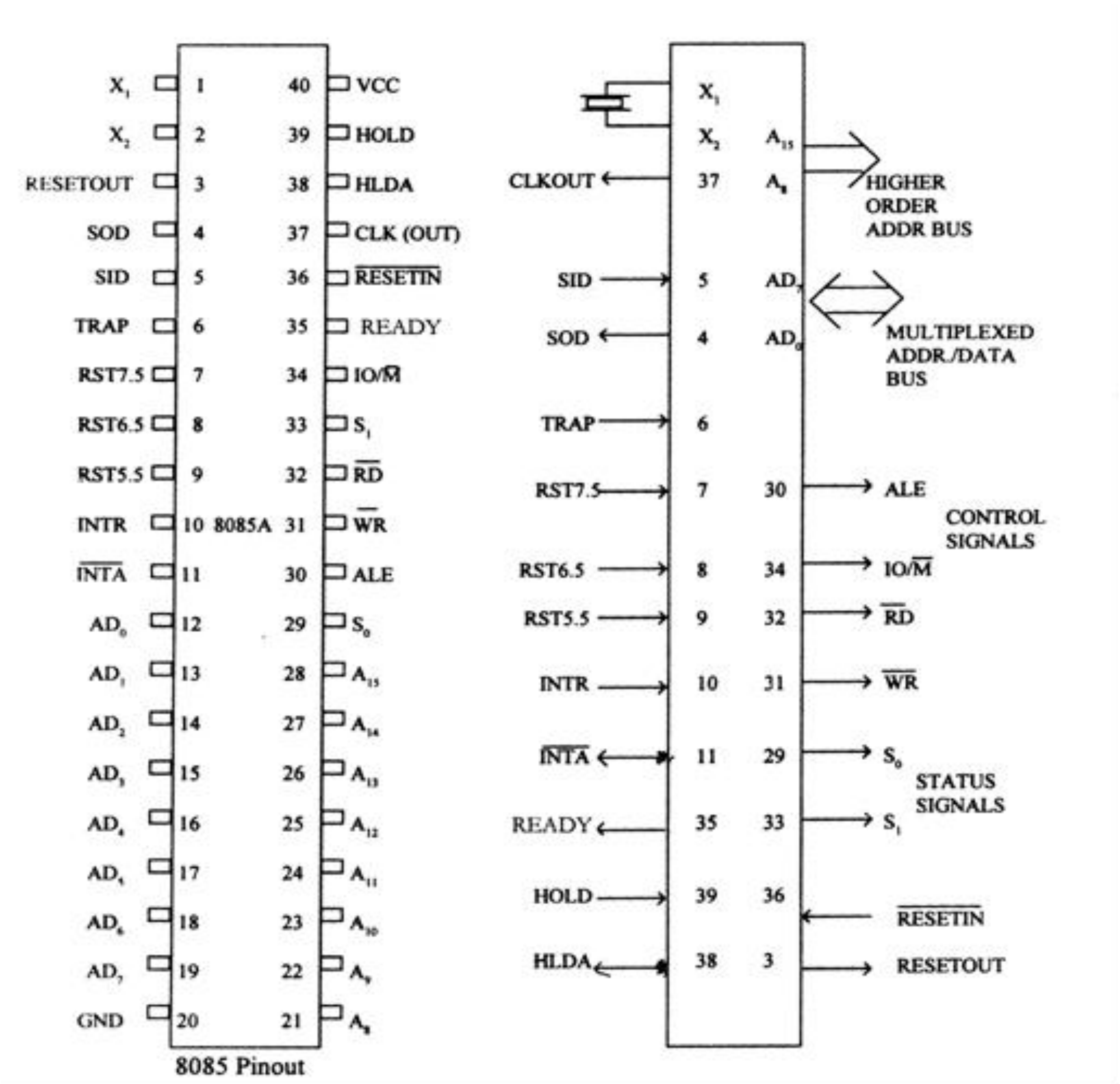
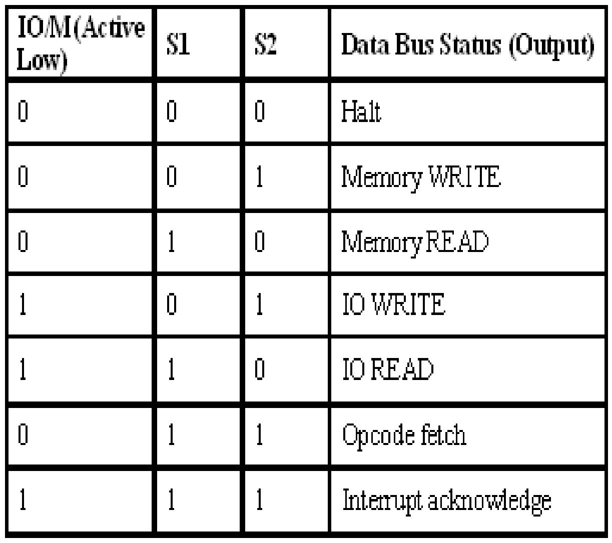


Fig 1.1(a) - Pin Diagram of 8085 & Fig.1.1(b) - logical schematic of Pin diagram.



IO/M(Active Low)	S1	S2	Data Bus Status (Output)
0	0	0	Halt
0	0	1	Memory WRITE
0	1	0	Memory READ
1	0	1	IO WRITE
1	1	0	IO READ
0	1	1	Opcode fetch
1	1	1	Interrupt acknowledge

(vi). Interrupts and Externally initiated operations:

- They are the signals initiated by an external device to request the microprocessor to do a particular task or work.
- There are five hardware interrupts called,

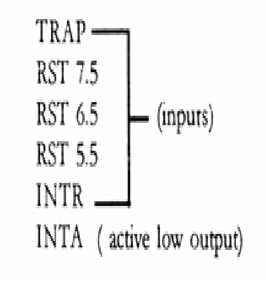


Fig. 1,2

On receipt of an interrupt, the microprocessor acknowledges the interrupt by the active low INTA (Interrupt Acknowledge) signal.

READY (input)

- Memory and I/O devices will have slower response compared to microprocessors.
- Before completing the present job such a slow peripheral may not be able to handle further data or control signal from CPU.
- The processor sets the READY signal after completing the present job to access the data.
- The microprocessor enters into WAIT state while the READY pin is disabled.

Direct Memory Access (DMA):

Tri state devices:

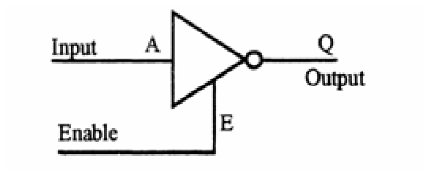


Fig. 1.3

- 3 output states are high & low states and additionally a high impedance state. When enable E is high the gate is enabled and the output Q can be 1 or 0 (ifA is 0, Q is 1, otherwise Q is 0). However, when E is low the gate is disabled and the output Q entersinto a high impedance state

E	A	Q	State
1(high)	0	1	High
1	1	0	Low
0(low)	0	0	High impedance
0	1	0	High impedance

.

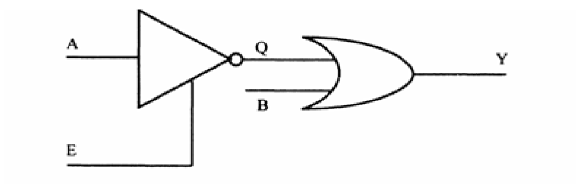


Fig. 1.4

- For both high and low states, the output Q draws a current from the input of the OR gate.
- When E is low, Q enters a high impedance state; high impedance means it is electrically isolated from the OR gate's input, though it is physically connected. Therefore, it does not draw any current from the OR gate's input.
- When 2 or more devices are connected to a common bus, to prevent the devices from interfering with each other, the tristate gates are used to disconnect all devices except the one that is communicating at a given instant.
- The CPU controls the data transfer operation between memory and I/O device. Direct Memory Access operation is used for large volume data transfer between memory and an I/O device directly.
- The CPU is disabled by tri-stating its buses and the transfer is effected directly by external control circuits.
- HOLD signal is generated by the DMA controller circuit. On receipt of this signal, the microprocessor acknowledges the request by sending out HLDA signal and leaves out the control of the buses. After the HLDA signal the DMA controller starts the direct transfer of data.

(vi). Single Bit Serial I/O ports:

- SID (input) - Serial input data line
- SOD (output) - Serial output data line
- These signals are used for serial communication.

1.4. Bus Structure of 8085 Microprocessor : There are three buses in Microprocessor:

1. Address Bus 2. Data Bus . 3. Control Bus

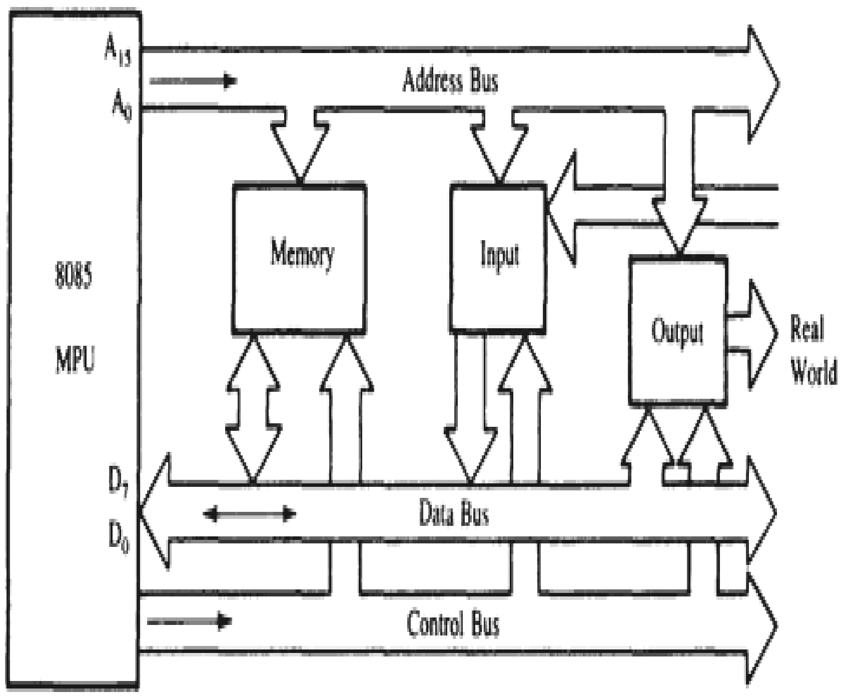


Fig. 1.5: Bus Structure

1.**Address Bus**:- Genearlly, Microprocessor has 16 bit address bus. The bus over which the CPU sends out the address of the memory location is known as Address bus. The address bus carries the address of memory location to be written or to be read from. The address bus is unidirectional. It means bits flowing occurs only in one direction, only from microprocessor to peripheral devices.

2. Data Bus:- 8085 Microprocessor has 8 bit data bus. So it can be used to carry the 8 bit data starting from 00000000H(00H) to 11111111H(FFH). Here 'H' tells the Hexadecimal Number. It is bidirectional. These lines are used for data flowing in both direction means data can be transferred or can be received through these lines. The data bus also connects the I/O ports and CPU. The largest number that can appear on the data bus is 11111111.

3.Control Bus:-The control bus is used for sending control signals to the memory and I/O devices. The CPU sends control signal on the control bus to enable the outputs of addressed memory devices or I/O port devices. Some of the control bus signals are as follows:

(i).Memory read (ii) . Memory write (iii). I/O read (iv). I/O write.

1.5. Architecture of 8085 Microprocessor :

The Functional Block Diagram of 8085 Microprocessor is given below:

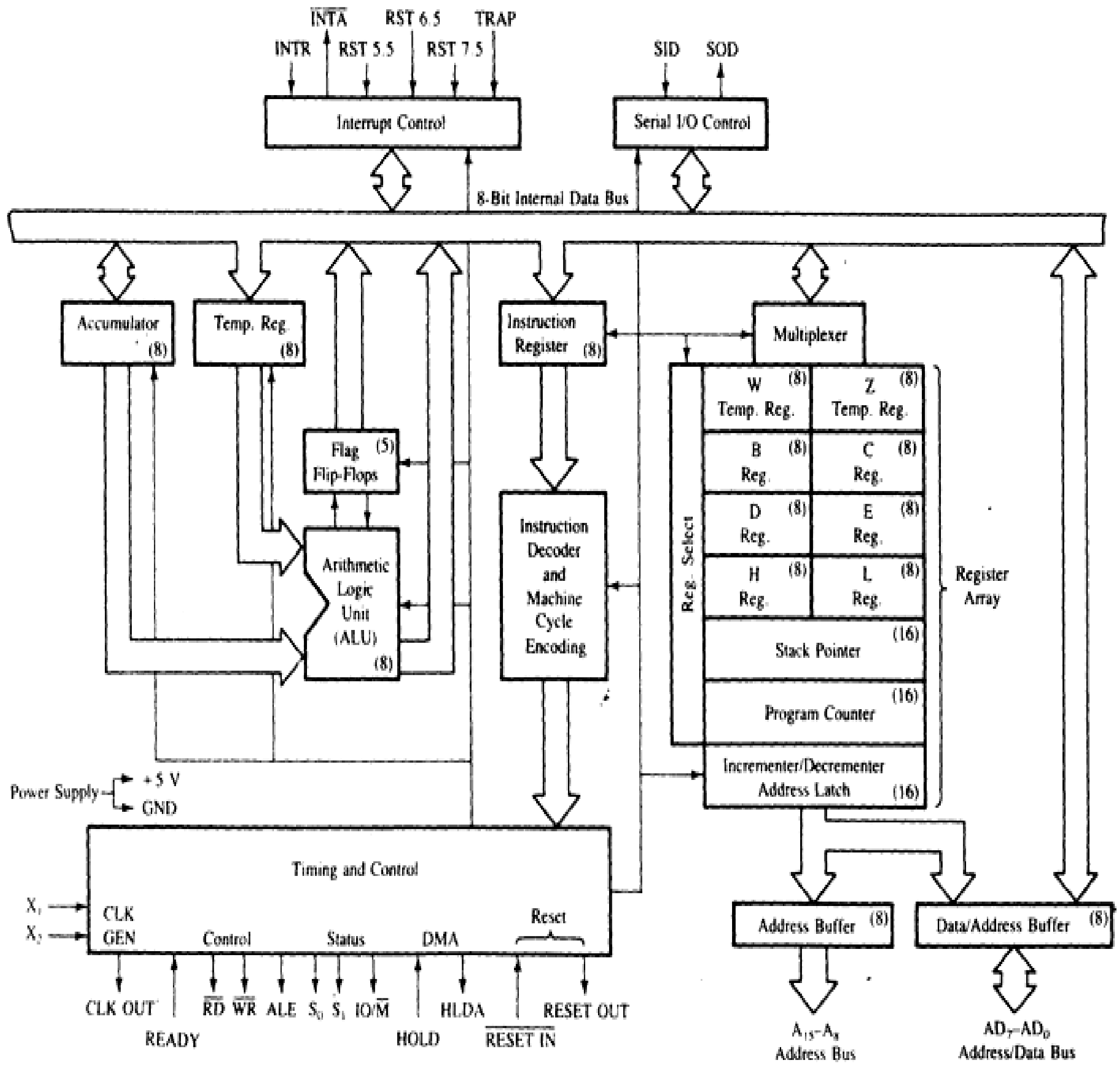


Fig. 1.6 Architecture of 8085

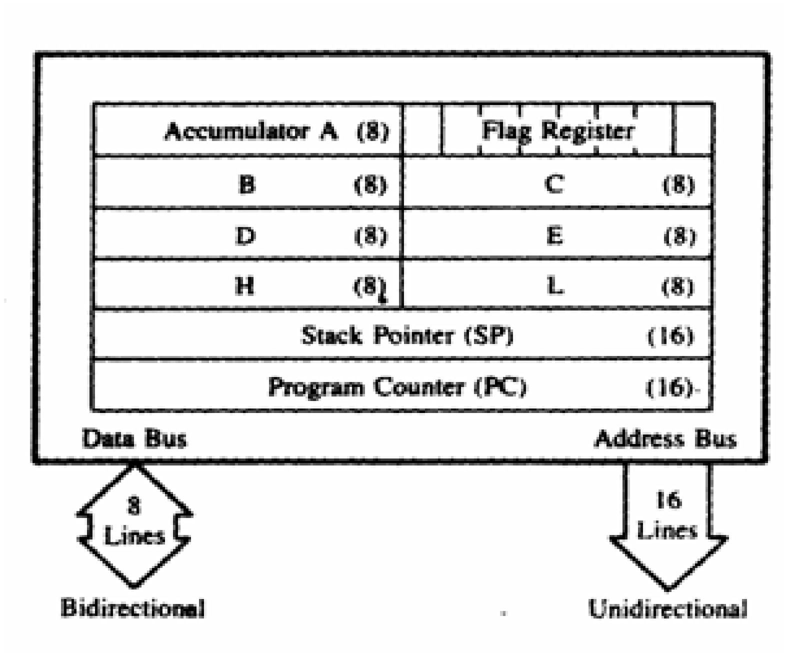


Fig.1.7

Acumulator:- It is a 8-bit register which is used to perform airthmetical and logical operation. It stores the output of any operation. It also works as registers for i/o accesses.

Temporary Register:- It is a 8-bit register which is used to hold the data on which the acumulator is computing operation. It is also called as operand register because it provides operands to ALU.

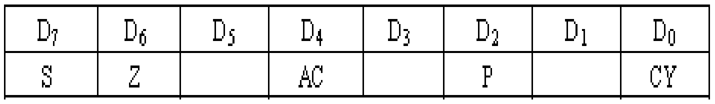
Registers:- These are general purposes registers. Microprocessor consists 6 general purpose registers of 8-bit each named as B, C, D, E,H and L. Generally theses registers are not used for storing the data permanently. It carries the 8-bits data. These are used only during the execution of the instructions. These registers can also be used to carry the 16 bits data by making the pair of 2 registers. The valid register pairs available are BC, DE HL. We can not use the pairs except BC, DE and HL. These registers are programmed by user.

A**LU:-**ALU performs the airthmetic operations and logical operation.

Flag Registers:-It consists of 5 flip flop which changes its status according to the result stored in an accumulator. It is also known as status registers. It is connected to the ALU. There are five flip-flops in the flag register are as follows:

1.Sign(S) 2.Zero (Z) 3.Auxiliary carry (AC) 4.Parity (P) 5.Carry (C)

The bit position of the flip flop in flag register is:



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

All of the three flip flop set and reset according to the stored result in the accumulator.

1. Sign- If D7 of the result is 1 then sign flag is set otherwise reset. As we know that a number on the D7 always desides the sign of the number.
if D7 is 1: the number is negative.
if D7 is 0: the number is positive.

2. Zeros (Z)-If the result stored in an accumulator is zero then this flip flop is set otherwise it is reset.
3.Auxiliary carry(AC)-If any carry goes from D3 to D4 in the output then it is set otherwise it is reset.
4.Parity(P)-If the no of 1's is even in the output stored in the accumulator then it is set otherwise it is reset for the odd.
5.Carry(C)-If the result stored in an accumulator generates a carry in its final output then it is set otherwise it is reset.

Instruction registers(IR):-It is a 8-bit register. When an instruction is fetched from memory then it is stored in this register.

Instruction Decoder:- Instruction decoder identifies the instructions. It takes the informations from instruction register and decodes the instruction to be performed.

Program Counter:-It is a 16 bit register used as memory pointer. It stores the memory address of the next instruction to be executed. So we can say that this register is used to sequencing the program. Generally the memory have 16 bit addresses so that it has 16 bit memory.
The program counter is set to 0000H.

Stack Pointer:-It is also a 16 bit register used as memory pointer. It points to the memory location called stack. Generally stack is a reserved portion of memory where information can be stores or taken back together.

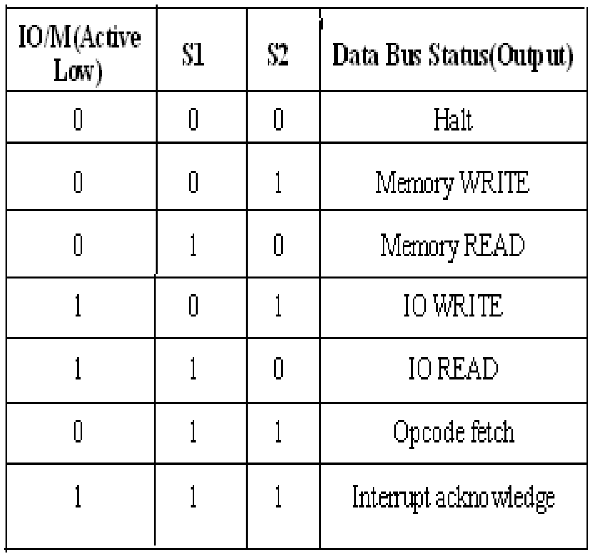
Timing and Control Unit:-It provides timing and control signal to the microprocessor to perform the various operation.It has three control signal. It controls all external and internal circuits. It operates with reference to clock signal.It synchronizes all the data transfers.

There are three control signal:

1.ALE-Airthmetic Latch Enable, It provides control signal to synchronize the components of microprocessor.

2.RD- This is active low used for reading operation.

3.WR-This is active low used for writing operation. There are three status signal used in microprocessor S0, S1 and IO/M. It changes its status according the provided input to these pins.



IO/M(Active Low)	S1	S2	Data Bus Status(Output)
0	0	0	Halt
0	0	1	Memory WRITE
0	1	0	Memory READ
1	0	1	IO WRITE
1	1	0	IO READ
0	1	1	Opcode fetch
1	1	1	Interrupt acknowledge

Serial Input Output Control-There are two pins in this unit. This unit is used for serial data communication.

Interrupt Unit-There are 6 interrupt pins in this unit. Generally an external hardware is connected to these pins. These pins provide interrupt signal sent by external hardware to microprocessor and microprocessor sends acknowledgement for receiving the interrupt signal. Generally INTA is used for acknowledgement.

8085- Registers

The 8085 has six general purpose registers to store 8 bit data. These are identified as B, C, D, E, H, L. they can be combined as register pairs BC, DE, and HL, to perform 16 bit operations.

Accumulator

The acc is an 8 bit register that is part of the arithmetic logic unit [ALU]. This register is used to store 8 bit data and to perform arithmetic and logical operations. The result of the operation is stored in the accumulator and identified as A.

Flags

The arithmetic logic unit [ALU] includes 5 flip flops which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called zero (Z), carry (CY), sign(S), parity (P), and auxiliary carry (AC). The microprocessor used these flags to test data conditions.

Program counter

The microprocessor uses the PC register to sequence the execution of the instructions. The function of the PC is to point to the memory address from which the next byte is to be fetched. When a byte is being fetched, the pc is increased by one to point to the next memory location.

Stack pointer

The SP is also a 16 bit register used as a memory pointer. It points to a memory location in R/W memory, called the *stack*

Interrupts In 8085

What is Interrupt?

Interrupt is a mechanism by which an I/O or an instruction can suspend the normal execution of processor and get itself serviced. Generally, a particular task is assigned to that interrupt signal. In the microprocessor based system the interrupts are used for data transfer between the peripheral devices and the microprocessor.

Interrupt Service Routine(ISR)

Interrupt means to break the sequence of operation. While the CPU is executing a program an interrupt breaks the normal sequence of execution of instructions & diverts its execution to some other program. This program to which the control is transferred is called the *interrupt service routine.* A small program or a routine that when executed services the corresponding interrupting source is called as an ISR.

Execution of Interrupts

When there is an interrupt requests to the Microprocessor then after accepting the interrupts Microprocessor send the INTA (active low) signal to the peripheral. The vectored address of particular interrupt is stored in program counter. The processor executes an interrupt service routine (ISR) addressed in program counter.

There are two types of interrupts used in 8085 Microprocessor:

Hardware Interrupts and Software Interrupts

Software Interrupts

A software interrupts is a particular instructions that can be inserted into the desired location in the rpogram. There are eight Software interrupts in 8085 Microprocessor. From RST0 to RST7.

RST0, RST1, RST2, RST3, RST4, RST5, RST6, RST7

They allow the microprocessor to transfer program control from the main program to the subroutine program. After completing the subroutine program, the program control returns back to the main program.

Hardware Interrupts

There are 6 interrupt pins in the microprocessor used as Hardware Interrrupts given below:

TRAP, RST7.5, RST6.5, RST5.5, INTR

Note:

INTA is not an interrupt. INTA is used by the Microprocessor for sending the acknowledgement. TRAP has highest priority and RST7.5 has second highest priority and so on.

TRAP

It is non maskable edge and level triggered interrupt. TRAP has the highest priority and vectored interrupt. Edge and level triggered means that the TRAP must go high and remain high until it is acknowledged. In case of sudden power failure, it executes a ISR and send the data from main memory to backup memory.

As we know that TRAP can not be masked but it can be delayed using HOLD signal. This interrupt transfers the microprocessor's control to location 0024H.

TRAP interrupts can only be masked by reseting the microprocessor. There is no other way to mask it.

RST7.5

It has the second highest priority. It is maskable and edge level triggered interrupt. The vector address of this interrupt is 003CH. Edge sensitive means input goes high and no need to maintain high state until it is recognized. It can also be reset or masked by reseting microprocessor. It can also be resetted by DI instruction.

RST6.5 and RST5.5

These are level triggered and maskable interrupts. When RST6.5 pin is at logic 1, INTE flip-flop is set. RST 6.5 has third highest priority and RST 5.5 has fourth highest priority.

It can be masked by giving DI and SIM instructions or by reseting microprocessor.

INTR

It is level triggered and maskable interrupt. The following sequence of events occurs when INTR signal goes high. The 8085 checks the status of INTR signal during execution of each instruction. If INTR signal is high, then 8085 complete its current instruction and sends active low interrupt acknowledge signal, if the interrupt is enabled. On receiving the instruction, the 8085 save the address of next instruction on stack and execute received instruction. It has the lowest priority. It can be disabled by reseting the microprocessor or by DI and SIM instruction.

1.6. Intel 8086 Microprocessor:

Pin Diagram:

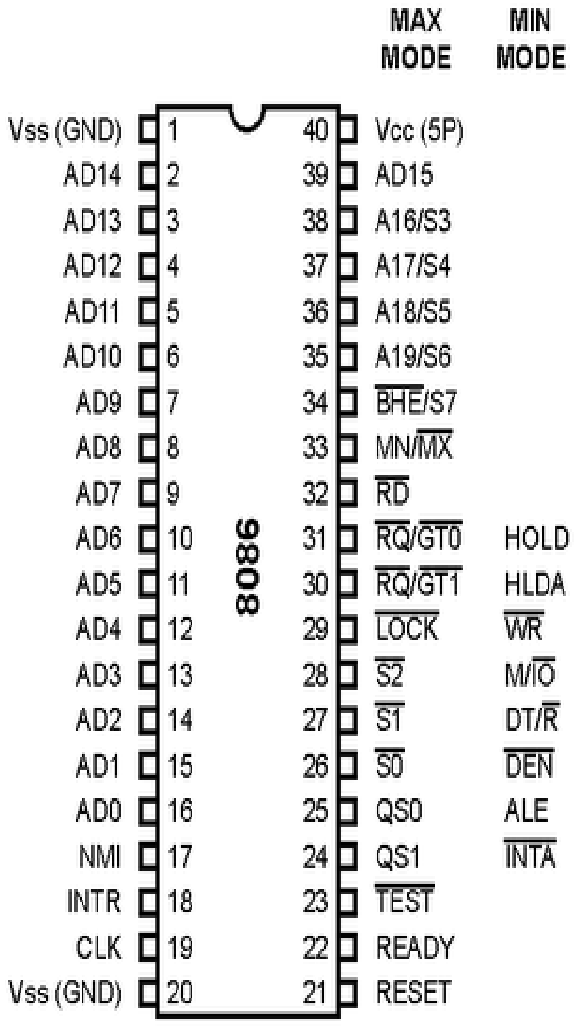


Fig. 1.8 Pin diagram of 8086 micprocessor

The 8086 can operate in two modes these are the minimum mode and maximum mode . For minimum mode, a unique processor system with a single 8086 and for Maximum mode a multi processor system with more than one 8086. The following pin function descriptions are for the microprocessor 8086 in either minimum or maximum mode.

AD0 - AD15 (I/O): Address Data Bus

These lines constitute the time multiplexed memory/IO address during the first clock cycle (T1) and data during T2, T3 and T4 clock cycles. A0 is analogous to BHE for the lower byte of the data bus, pins D0-D7. A0 bit is Low during T1 state when a byte is to be transferred on the lower portion of the bus in memory or I/O operations. 8-bit oriented devices tied to the lower half would normally use A0 to condition chip select functions. These lines are active high and float to tri-state during interrupt acknowledge and local bus "Hold acknowledge".

A19/S6, A18/S5, A17/S4, A16/S3 (0): Address/Status

During T1 state these lines are the four most significant address lines for memory operations. During I/O operations these lines are low. During memory and I/O operations, status information is available on these lines during T2, T3, and T4 states.S5: The status of the interrupt enable flag bit is updated at the beginning of each cycle. The status of the flag is indicated through this bus.
S6: When Low, it indicates that 8086 is in control of the bus. During a "Hold acknowledge" clock period, the 8086 tri-states the S6 pin and thus allows another bus master to take control of the status bus.

S3 & S4:

Lines are decoded as follows:

A17/S4	**A16/S3**	**Function**
0	0	Extra segment access
0	1	Stack segment access
1	0	Code segment access
1	1	Data segment access

After the first clock cycle of an instruction execution, the A17/S4 and A16/S3 pins specify which segment register generates the segment portion of the 8086 address. Thus by decoding these lines and using the decoder outputs as chip selects for memory chips, up to 4 Megabytes (one Mega per segment) of memory can be accesses. This feature also provides a degree of protection by preventing write operations to one segment from erroneously overlapping into another segment and destroying information in that segment.

BHE /S7 (O): Bus High Enable/Status: During T1 state the BHE should be used to enable data onto the most significant half of the data bus, pins D15 - D8. Eight-bit oriented devices tied to the upper half of the bus would normally use BHE to control chip select functions. BHE is Low during T1 state of read, write and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus.

The S7 status information is available during T2, T3 and T4 states. The signal is active Low and floats to 3-state during "hold" state. This pin is Low during T1 state for the first interrupt acknowledge cycle.

RD (O): READ: The Read strobe indicates that the processor is performing a memory or I/O read cycle. This signal is active low during T2 and T3 states and the Tw states of any read cycle. This signal floats to tri-state in "hold acknowledge cycle".

TEST (I): TEST pin is examined by the "WAIT" instruction. If the TEST pin is Low, execution continues. Otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.

INTR (I): Interrupt Request: It is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector look up table located in system memory. It can be internally masked by software resetting the interrupt enable bit INTR is internally synchronized. This signal is active HIGH.

NMI (I): Non-Maskable Interrupt: An edge triggered input, causes a type-2 interrupt. A subroutine is vectored to via the interrupt vector look up table located in system memory. NMI is not maskable internally by software. A transition from a LOW to HIGH on this pin initiates the interrupt at the end of the current instruction. This input is internally synchronized.

Reset (I): Reset causes the processor to immediately terminate its present activity. To be recognised, the signal must be active high for at least four clock cycles, except after power-on which requires a 50 Micro Sec. pulse. It causes the 8086 to initialize registers DS, SS, ES, IP and flags to all zeros. It also initializes CS to FFFF H. Upon removal of the RESET signal from the RESET pin, the 8086 will fetch its next instruction from the 20 bit physical address FFFF0H. The reset signal to 8086 can be generated by the 8284. (Clock generation chip). To guarantee reset from power-up, the reset input must remain below 1.5 volts for 50 Micro sec. after Vcc has reached the minimum supply voltage of 4.5V.

Ready (I): Ready is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory or I/O is synchronized by the 8284 clock generator to form READY. This signal is active HIGH. The 8086 READY input is not synchronized. Correct operation is not guaranteed if the setup and hold times are not met.

CLK (I): Clock: Clock provides the basic timing for the processor and bus controller. It is asymmetric with 33% duty cycle to provide optimized internal timing. Minimum frequency of 2 MHz is required, since the design of 8086 processors incorporates dynamic cells. The maximum clock frequencies of the 8086-4, 8086 and 8086-2 are4MHz, 5MHz and 8MHz respectively. Since the 8086 does not have on-chip clock generation circuitry, and 8284 clock generator chip must be connected to the 8086 clock pin. The crystal connected to 8284 must have a frequency 3 times the 8086 internal frequency. The 8284 clock generation chip is used to generate READY, RESET and CLK.

MN/MX (I): Maximum / Minimum:

This pin indicates what mode the processor is to operate in. In minimum mode, the 8086 itself generates all bus control signals. In maximum mode the three status signals are to be decoded to generate all the bus control signals.

Minimum Mode Pins : The following 8 pins function descriptions are for the 8086 in minimum mode; MN/ MX = 1. The corresponding 8 pins function descriptions for maximum mode is explained later.
(i) M/IO (O): Status line: This pin is used to distinguish a memory access or an I/O accesses. When this pin is Low, it accesses I/O and when high it access memory. M / IO becomes valid in the T4 state preceding a bus cycle and remains valid until the final T4 of the cycle. M/IO floats to 3 - state OFF during local bus "hold acknowledge".

(ii) WR (O): Write: Indicates that the processor is performing a write memory or write IO cycle, depending on the state of the M /IOsignal. WR is active for T2, T3 and Tw of any write cycle. It is active LOW, and floats to 3-state OFF during local bus "hold acknowledge ".

(iii) INTA (O): Interrupt Acknowledge: It is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T2, T3, and T4 of each interrupt acknowledge cycle.

(iv) ALE (O): Address Latch Enable: ALE is provided by the processor to latch the address into the 8282/8283 address latch. It is an active high pulse during T1 of any bus cycle. ALE signal is never floated.

(v) DT/ R (O): DATA Transmit/Receive: In minimum mode, 8286/8287 transceiver is used for the data bus. DT/ R is used to control the direction of data flow through the transceiver. This signal floats to tri-state off during local bus "hold acknowledge".

(vi) DEN (O): Data Enable: It is provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. DEN is active LOW during each memory and IO access. It will be low beginning with T2 until the middle of T4, while for a write cycle, it is active from the beginning of T2 until the middle of T4. It floats to tri-state off during local bus "hold acknowledge"**.**

(vii) HOLD & HLDA (I/O): Hold and Hold Acknowledge: Hold indicates that another master is requesting a local bus "HOLD". To be acknowledged, HOLD must be active HIGH. The processor receiving the "HOLD " request will issue HLDA (HIGH) as an acknowledgement in the middle of the T1-clock cycle. Simultaneous with the issue of HLDA, the processor will float the local bus and control lines. After "HOLD" is detected as being Low, the processor will lower the HLDA and when the processor needs to run another cycle, it will again drive the local bus and control lines.

Maximum Mode : The following pins function descriptions are for the 8086/8088 systems in maximum mode (i.e.. MN/MX = 0). Only the pins which are unique to maximum mode are described below.

(i) S2, S1, S0 (O): Status Pins: These pins are active during T4, T1 and T2 states and is returned to passive state (1,1,1 during T3 or Tw (when ready is inactive). These are used by the 8288 bus controller to generate all memory and I/O operation) access control signals. Any change by S2, S1, S0 during T4 is used to indicate the beginning of a bus cycle. These status lines are encoded as shown in table 1.1.

Table 1.1:

S2	**S1**	**S0**	**Chracteristics**
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code access 1 0 1 Read memory
1	1	0	Write memory
1	1	1	Passive state

(ii) QS0, QS1 (O): Queue – Status: Queue Status is valid during the clock cycle after which the queue operation is performed. QS0, QS1 provide status to allow external tracking of the internal 8086 instruction queue. The condition of queue status is shown in table 1.2. Queue status allows external devices like In-circuit Emulators or special instruction set extension co-processors to track the CPU instruction execution. Since instructions are executed from the 8086 internal queue, the queue status is presented each CPU clock cycle and is not related to the bus cycle activity. This mechanism allows (1) A processor to detect execution of a ESCAPE instruction which directs the co- processor to perform a specific task and (2) An in-circuit Emulator to trap execution of a specific memory location.

Table:1.2

QS0	QS1	Chracteristics
0	0	No operation
0	1	First byte of opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

(iii) LOCK (O): It indicates to another system bus master, not to gain control of the system bus while LOCK is active Low. The LOCK signal is activated by the "LOCK" prefix instruction and remains active until the completion of the instruction. This signal is active Low and floats to tri-state OFF during 'hold acknowledge'.

Example: LOCK XCHG reg., Memory ; Register is any register and memory GT0
; is the address of the semaphore.

(iv) RQ/GT0 and RQ/GT1 (I/O): Request/Grant: These pins are used by other processors in a multi processor organization. Local bus masters of other processors force the processor to release the local bus at the end of the processors current bus cycle. Each pin is bi-directional and has an internal pull up resistors. Hence they may be left un-connected.

1.7. General purpose registers

8086 CPU has 8 general purpose registers, each register has its own name:

- **AX** - the accumulator register (divided into **AH / AL**):
    - Generates shortest machine code
    - Arithmetic, logic and data transfer
    - One number must be in AL or AX
    - Multiplication & Division
    - Input & Output
- **BX** - the base address register (divided into **BH / BL**).

- **CX** - the count register (divided into **CH / CL**):
    - Iterative code segments using the LOOP instruction
    - Repetitive operations on strings with the REP command
    - Count (in CL) of bits to shift and rotate
- **DX** - the data register (divided into **DH / DL**):

DX:AX concatenated into 32-bit register for some MUL and DIV operations
Specifying ports in some IN and OUT operations

- **SI** - source index register:
    - Can be used for pointer addressing of data
    - Used as source in some string processing instructions
    - Offset address relative to DS
- **DI** - destination index register: Can be used for pointer addressing of data used as destination in some string processing instructions.

Offset address relative to ES

- **BP** - base pointer:
    - Primarily used to access parameters passed via the stack
    - Offset address relative to SS
- **SP** - stack pointer:
    - Always points to top item on the stack
    - Offset address relative to SS
    - Always points to word (byte at even address)
    - An empty stack will had SP = FFFEh

1.8. Segment registers

- **CS** - points at the segment containing the current program.
- **DS** - generally points at segment where variables are defined.
- **ES** - extra segment register, it's up to a coder to define its usage.
- **SS** - points at the segment containing the stack.

Although it is possible to store any data in the segment registers, this is never a good idea. The segment registers have a very special purpose - pointing at accessible blocks of memory.

Segment registers work together with general purpose register to access any memory value. For example if we would like to access memory at the physical address **12345h** (hexadecimal), we should set the **DS = 1230h** and **SI = 0045h**. This is good, since this way we can access much more memory than with a single register that is limited to 16 bit values. CPU makes a calculation of physical address by multiplying the segment register by 10h and adding general purpose register to it (1230h * 10h + 45h = 12345h): The address formed with 2 registers is called an **effective address**. By default **BX, SI** and **DI** registers work with **DS** segment register; **BP** and **SP** work with **SS** segment register. Other general purpose registers cannot form an effective address. Also, although **BX** can form an effective address, **BH** and **BL** cannot.

1.9. Special purpose registers

- **IP** - the instruction pointer:
    - Always points to next instruction to be executed
    - Offset address relative to CS

IP register always works together with **CS** segment register and it points to currently executing instruction.

1.10. Register

- **Flags Register** - determines the current state of the processor.

  Flags Register is modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program. Generally we cannot access these registers directly.

- **Carry Flag (CF)** - this flag is set to **1** when there is an unsigned overflow. For example when you add bytes 255 + 1 (result is not in range 0...255). When there is no overflow this flag is set to 0.
- **Parity Flag (PF)** - this flag is set to 1 when there is even number of one bits in result, and to 0 when there is odd number of one bits. Even if result is a word only 8 low bits are analyzed.
- **Auxiliary Flag (AF)** - set to 1 when there is an unsigned overflow for low nibble (4 bits).
- **Zero Flag (ZF)** - set to 1 when result is zero. For none zero result this flag is set to 0.
- **Sign Flag (SF)** - set to 1 when result is negative. When result is positive it is set to 0. Actually this flag take the value of the most significant bit.
- **Trap Flag (TF)** - Used for on-chip debugging.
- **Interrupt enable Flag (IF)** - when this flag is set to 1 CPU reacts to interrupts from external devices.
- **Direction Flag (DF)** - this flag is used by some instructions to process data chains, when this flag is set to 0 - the processing is done forward, when this flag is set to 1 the processing is done backward.

- **Overflow Flag (OF)** - set to 1 when there is a signed overflow. For example, when you add bytes 100 + 50 (result is not in range -128...127).

- **Flags Instructions**
    - CLI - Clear Interrupt Flag
    - CLD - Clear Direction Flag
    - CLC - Clear Carry Flag
    - STC - Set Interrupt Flag
    - STD - Set Direction Flag
    - STC - Set Carry Flag

1.11. Interrupts of 8086

There are two main types of interrupt in the 8086 microprocessor, internal and external hardware interrupts. Hardware interrupts occur when a peripheral device asserts an interrupt input pin of the microprocessor. Whereas internal interrupts are initiated by the state of the CPU (e.g. divide by zero error) or by an instruction.

Provided the interrupt is permitted, it will be acknowledged by the processor at the end of the current memory cycle. The processor then services the interrupt by branching to a special service routine written to handle that particular interrupt. Upon servicing the device, the processor is then instructed to continue with what is was doing previously by use of the "return from interrupt" instruction.

The status of the programme being executed must first be saved. The processors registers will be saved on the stack, or, at very least, the programme counter will be saved. Preserving those registers which are not saved will be the responsibility of the interrupt service routine. Once the programme counter has been saved, the processor will branch to the address of the service routine.

(i) Edge or Level sensitive Interrupts

Edge level interrupts are recognised on the falling or rising edge of the input signal. They are generally used for high priority interrupts and are latched internally inside the processor. If this latching was not done, the processor could easily miss the falling edge (due to its short duration) and thus not respond to the interrupt request.

Level sensitive interrupts overcome the problem of latching, in that the requesting device holds the interrupt line at a specified logic state (normally logic zero) till the processor acknowledges the interrupt. This type of interrupt can be shared by other devices in a wired 'OR' configuration, which is commonly used to support daisy chaining and other techniques.

(ii) Maskable Interrupts

The processor can inhibit certain types of interrupts by use of a special interrupt mask bit. This mask bit is part of the flags/condition code register, or a special interrupt register. In the 8086 microprocessor if this bit is clear, and an interrupt request occurs on the Interrupt Request input, it is ignored.

(iii) Non-Maskable Interrupts

There are some interrupts which cannot be masked out or ignored by the processor. These are associated with high priority tasks which cannot be ignored (like memory parity or bus faults). In general, most processors support the Non-Maskable Interrupt (NMI). This interrupt has absolute priority, and when it occurs, the processor will finish the current memory cycle, then branch to a special routine written to handle the interrupt request.

(iv) Advantages of Interrupts

Interrupts are used to ensure adequate service response times by the processing. Sometimes, with software polling routines, service times by the processor cannot be guaranteed, and data may be lost. The use of interrupts guarantees that the processor will service the request within a specified time period, reducing the likelihood of lost data.

(v) Interrupt Latency

The time interval from when the interrupt is first asserted to the time the CPU recognises it. This will depend much upon whether interrupts are disabled, prioritized and what the processor is currently executing. At times, a processor might ignore requests whilst executing some indivisible instruction stream (read-write-modify cycle). The figure that matters most is the longest possible interrupt latency time.

(vi) Interrupt Response Time

The time interval between the CPU recognising the interrupt to the time when the first instruction of the interrupt service routine is executed. This is determined by the processor architecture and clock speed.

UNIT-II

Instruction set of 8085 and Assembly language Programming

2.1. Software:

Software is a collection of instructions that enable the user to interact with a computer, its hardware, or perform tasks. Without software, computers would be useless.

Examples and types of software: Below is a list of the different kinds of software

Software	*Example*
1. Antivirus	AVG, McAfee, Housecall
2. Email	Outlook, Thunderbird
3. Operating systems	Mac OS X10, Windows XP, Windows 7
4.Database	Accsee, MySQL, SQL
5. Audio/music program	iTunes, WinAmp
6. Spreadsheet	Excel
7. Wordprocessor	Word
8. Programming languages	C++, Java, HTML, VB, Fortran, Perl
9. Internet browser	Firefox, Google Chrome, Internet Explorer
10. Photo/Graphis program	Adobe Photoshop, CorelDraw

2.2. What is Assembly Language?

Assembly language is a low-level programming language for a computer. Assembly language is converted into executable machine code by a utility program referred to as an assembler like NASM, MASM, etc.

2.3 Advantages of Assembly Language

Having an understanding of assembly language makes one aware of −

- How programs interface with OS, processor, and BIOS;
- How data is represented in memory and other external devices;
- How the processor accesses and executes instruction;
- How instructions access and process data;
- How a program accesses external devices.

Other advantages of using assembly language are −

- It requires less memory and execution time;
- It allows hardware-specific complex jobs in an easier way;
- It is suitable for time-critical jobs;

- It is most suitable for writing interrupt service routines and other memory resident program.

2.3. Assembler:

An assembler program creates object code by translating combinations of mnemonics and syntax for operations and addressing modes into their numerical equivalents. This representation typically includes an *operation code* ("opcode") as well as other control bits and data. The assembler also calculates constant expressions and resolves symbolic names for memory locations and other entities. Most assemblers also include macro facilities for performing textual substitution – e.g., to generate common short sequences of instructions as inline, instead of *called* subroutines.

Some assemblers may also be able to perform some simple types of instruction set-specific optimizations. One concrete example of this may be the ubiquitous x86 assemblers from various vendors. Most of them are able to perform jump-instruction replacements (long jumps replaced by short or relative jumps) in any number of passes, on request.

Like early programming languages such as Fortran, Algol, Cobol and Lisp, assemblers have been available since the 1950s and the first generations of text based computer interfaces. There may be several assemblers with different syntax for a particular CPU or instruction set architecture. For instance, an instruction to add memory data to a register in a x86-family processor might be `add eax,[ebx]`, in original *Intel syntax*, whereas this would be written `addl (%ebx),%eax` in the *AT&T syntax* used by the GNU Assembler.

Number of passes

There are two types of assemblers based on how many passes through the source are needed to produce the executable program.

- One-pass assemblers go through the source code once. Any symbol used before it is defined will require "errata" at the end of the object code (or, at least, no earlier than the point where the symbol is defined) telling the linker or the loader to "go back" and overwrite a placeholder which had been left where the as yet undefined symbol was used.
- Multi-pass assemblers create a table with all symbols and their values in the first passes, then use the table in later passes to generate code.

In both cases, the assembler must be able to determine the size of each instruction on the initial passes in order to calculate the addresses of subsequent symbols. The original reason for the use of one-pass assemblers was speed of assembly – often a second pass would require rewinding and rereading the program source on tape or rereading a deck of cards or punched paper tape. With modern computers this has ceased to be an issue. The advantage of the multi-pass assembler is that the absence of errata makes the linking process faster.

High-level assemblers: More sophisticated high-level assemblers provide language abstractions such as:

- High-level procedure/function declarations and invocations
- Advanced control structures
- High-level abstract data types, including structures/records, unions, classes, and sets
- Sophisticated macro processing
- Object-oriented programming features such as classes, objects, abstraction, polymorphism, and inheritance.

2.4. Assembler directives:

Assembly directives, also called pseudo-opcodes, pseudo-operations or pseudo-ops, are commands given to an assembler "directing it to perform operations other than assembling instructions." Directives affect how the assembler operates and "may affect the object code, the symbol table, the listing file, and the values of internal assembler parameters." Sometimes the term pseudo-opcode is reserved for directives that generate object code, such as those that generate data. The names of pseudo-ops often start with a dot to distinguish them from machine instructions. Pseudo-ops can make the assembly of the program dependent on parameters input by a programmer, so that one program can be assembled different ways, perhaps for different applications. Or, a pseudo-op can be used to manipulate presentation of a program to make it easier to read and maintain. Another common use of pseudo-ops is to reserve storage areas for run-time data and optionally initialize their contents to known values.

Symbolic assemblers let programmers associate arbitrary names (*labels* or *symbols*) with memory locations and various constants. Usually, every constant and variable is given a name so instructions can reference those locations by name, thus promoting self-documenting code. In executable code, the name of each subroutine is associated with its entry point, so any calls to a subroutine can use its name. Inside subroutines, GOTO destinations are given labels. Some assemblers support *local symbols* which are lexically distinct from normal symbols (e.g., the use of "10$" as a GOTO destination).

Some assemblers, such as NASM provide flexible symbol management, letting programmers manage different namespaces, automatically calculate offsets within data structures, and assign labels that refer to literal values or the result of simple computations performed by the assembler. Labels can also be used to initialize constants and variables with relocatable addresses.

Assembly languages, like most other computer languages, allow comments to be added to program source code that will be ignored during assembly. Judicious commenting is essential in assembly language programs, as the meaning and purpose of a sequence of binary machine instructions can be difficult to determine. The "raw" (uncommented) assembly language generated by compilers or disassemblers is quite difficult to read when changes must be made.

2.5. The 8085 Addressing Modes :

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

(i). Immediate addressing.
(ii) Register addressing.
(iii) Direct addressing.
(iv) Indirect addressing.

(i) Immediate addressing:

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

Example: MVI R, data

(ii) Register addressing :

Data is provided through the registers. Example: MOV Rd, Rs

(iii) Direct addressing:

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

Example: IN 00H or OUT 01H

(iv) Indirect Addressing:

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

2.6. Instruction Set Classification:

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

(i) Data Transfer (Copy) Operations:

This group of instructions copy data from a location called a source to another location called a destination, without modifying the contents of the source. The various types of data transfer (copy) are listed below together with examples of each type:

Types	Examples
1. Between Registers.	1. Copy the contents of the register B into register D.
2. Specific data byte 32H to a register	2. Load register B with the data byte or a memory location. .
3. Between a memory location and a register.	3. From a memory location 2000H to register B.
4. Between an I/O device and the accumulator.	4. From an input keyboard to the accumulator.

(ii) Arithmetic Operations These instructions perform arithmetic operations such as addition, subtraction, increment, and decrement.

Addition - Any 8-bit number, or the contents of a register or the contents of a memory location can be added to the contents of the accumulator and the sum is stored in the accumulator. No two other 8-bit registers can be added directly (e.g., the contents of register B cannot be added directly to the contents of the register C). The instruction DAD is an exception; it adds 16-bit data directly in register pairs.

Subtraction – Any 8-bit number, or the contents of a register, or the contents of a memory location can be subtracted from the contents of the accumulator and the results stored in the accumulator. The subtraction is performed in 2's compliment, and the results if negative, are expressed in 2's complement. No two other registers can be subtracted directly.

Increment/Decrement - The 8-bit contents of a register or a memory location can be incremented or decrement by 1. Similarly, the 16-bit contents of a register pair (such as BC) can be incremented or decrementd by 1. These increment and decrement operations differ from addition and subtraction in an important way; i.e., they can be performed in any one of the registers or in a memory location.

(iii) Logical Operations : These instructions perform various logical operations with the contents of the accumulator. AND, OR, Exclusive-OR. Any 8-bit number, or the contents of a register, or of a memory location can be logically ANDed, Ored, or Exclusive-ORed with the contents of the accumulator. The results are stored in the ccumulator.

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

Compare- Any 8-bit number, or the contents of a register, or a memory location can be compared for equality, greater than, or less than, with the contents of the accumulator.

Complement - The contents of the accumulator can be complemented. All 0s are replaced by 1s and all 1s are replaced by 0s.

(iv) Branching Operations : This group of instructions alters the sequence of program execution either conditionally or unconditionally.

Jump - Conditional jumps are an important aspect of the decision-making process in the programming. These instructions test for a certain conditions (e.g., Zero or Carry flag) and alter the program sequence when the condition is met. In addition, the instruction set includes an instruction called unconditional jump.

Call, Return, and Restart - These instructions change the sequence of a program either by calling a subroutine or returning from a subroutine. The conditional Call and Return instructions also can test condition flags.

(v) Machine Control Operations : These instructions control machine functions such as Halt, Interrupt, or do nothing. The microprocessor operations related to data manipulation can be summarized in four functions:

1. copying data
2. performing arithmetic operations
3. performing logical operations
4. testing for a given condition and alerting the program sequence

Some important aspects of the instruction set are noted below:

1. In data transfer, the contents of the source are not destroyed; only the contents of the destination are changed. The data copy instructions do not affect the flags.

2. Arithmetic and Logical operations are performed with the contents of the accumulator, and the results are stored in the accumulator (with some expectations). The flags are affected according to the results.

3. Any register including the memory can be used for increment and decrement.

4. A program sequence can be changed either conditionally or by testing for a given data condition.

2.7. Instruction Format

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

Instruction word size

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

(i). One-word or 1-byte instructions
(ii). Two-word or 2-byte instructions
(iii). Three-word or 3-byte instructions

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

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

For example:

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

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

(ii) Two-Byte Instructions: In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode.

For example:

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

Assume that the data byte is 32H. The assembly language instruction is written as

Mnemonics	Hex code
MVI A, 32H	3E, 32H

(iii) Three-Byte Instructions : In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address. opcode + data byte + data byte

For example:

Task	Opcode	Operand	Hex Code
Transfer the program sequence to the memory location 2085H	JMP	2085H	C3- First byte 85- Second Byte 20- Third Byte

This instruction would require three memory locations to store in memory.
Three byte instructions - opcode + data byte + data byte

2.8. Clock cycle

The speed of a computer processor, or CPU, is determined by the clock cycle (Fig.2.1), which is the amount of time between two pulses of an oscillator. Generally speaking, the higher number of pulses per second, the faster the computer processor will be able to process information. The clock speed is measured in Hz, typically either megahertz (MHz) or gigahertz (GHz). For example, a 4GHz processor performs 4,000,000,000 clock cycles per second.

Computer processors can execute one or more instructions per clock cycle, depending on the type of processor. Early computer processors and slower processors can only execute one instruction per clock cycle, but faster, more advanced processors can execute multiple instructions per clock cycle, processing data more efficiently.

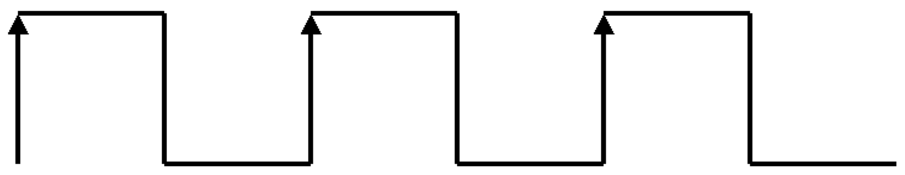


Fig.2.1.: Clock Signal

2.9. Machine cycle:

The steps performed by the computer processor for each machine language instruction received. The machine cycle is a 4 process cycle that includes reading and interpreting the machine language, executing the code and then storing that code.

Four steps of Machine cycle:

1. Fetch - Retrieve an instruction from the memory.
2. Decode - Tranlate the retrieved instruction into a series of computer commands.
3. Execute - Execute the computer commands.
4. Store – Save and write the results back in memory.

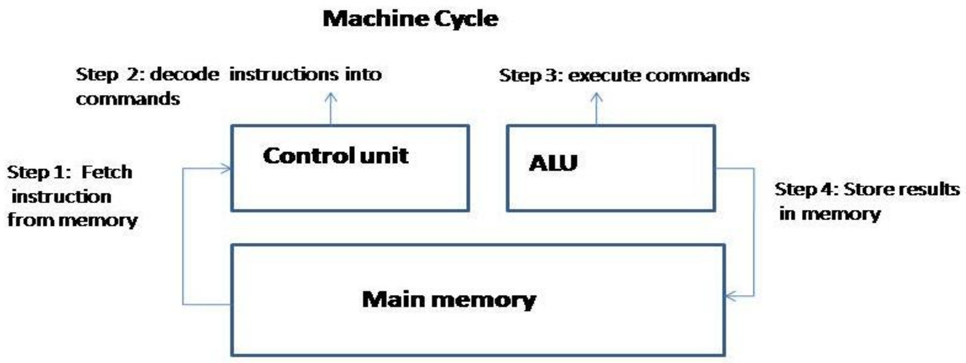
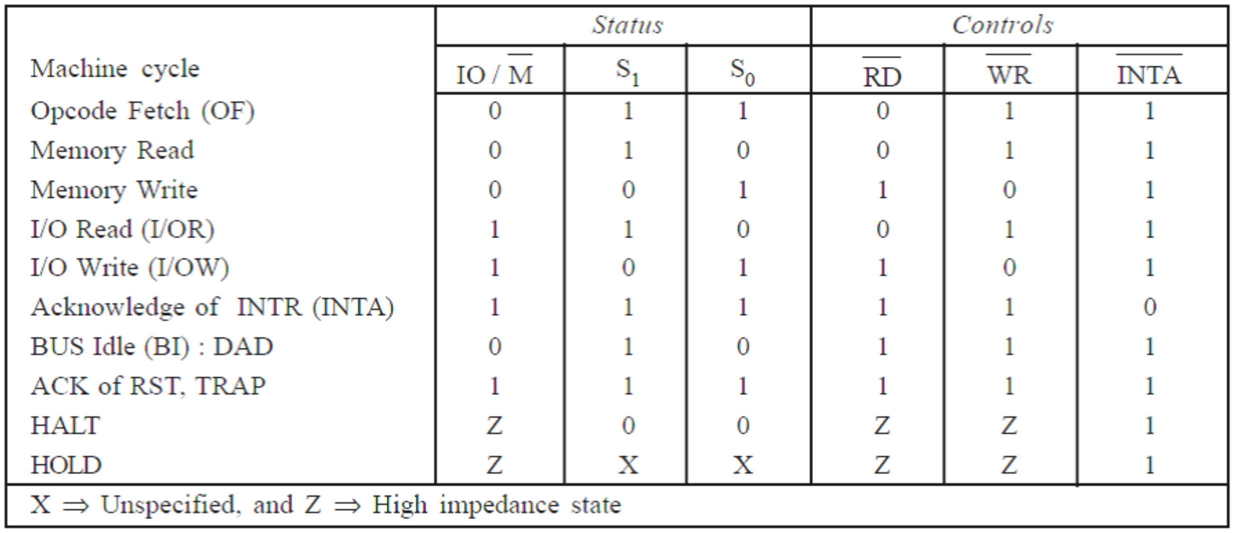

Machine Cycle
Step 2: decode instructions into commands
Step 3: execute commands
Step 1: Fetch instruction from memory
Control unit
ALU
Step 4: Store results in memory
Main memory

Fig.2.2 Machine cycle

Each machine cycle is composed of many clock cycle. 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 µP in doing so may take several cycles to perform fetch and execute operation. 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-2.1 shows details of the unique combination of these status signals to identify different machine cycles.



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

Table 2.1: Machine cycle status and control signals

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. 2.3. Each machine cycle consists of many clock periods/ cycles, called T-states. Each and every operation inside the microprocessor is under the control of the clock cycle. The clock signal determines the time taken by the microprocessor to execute any instruction. The clock cycle shown in Fig. 2.3 has two edges (leading and trailing or lagging). State is defined as the time interval between 2-trailing or leading edges of the clock. Machine cycle is the time required to transfer data to or from memory or I/O devices.

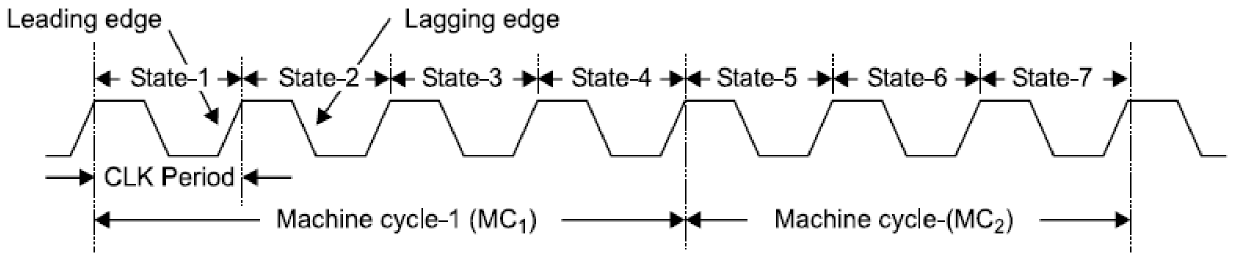


Fig.2.3 Machine cycles showing clock periods

2.10. Processor cycle - Instruction cycle

The function of the microprocessor is 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. Thus, 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

1. Fetch, and
2. 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. 2.4.

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

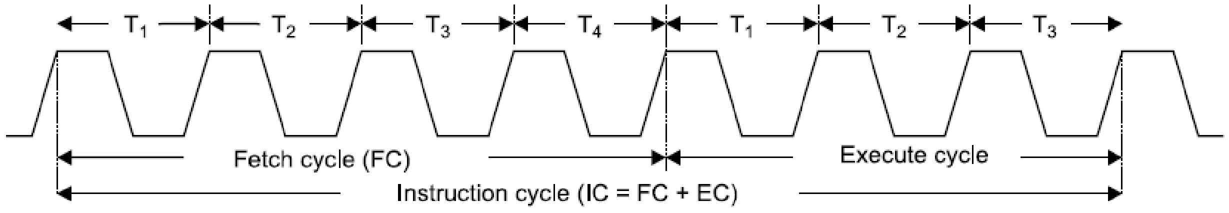
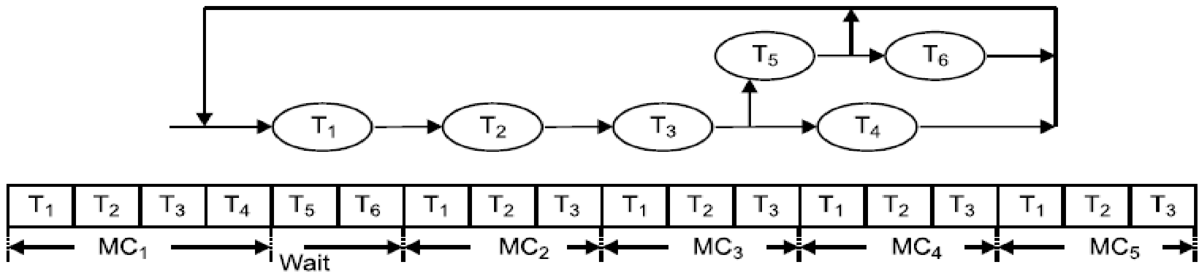


Fig.2.4 Processor Cycle

Instruction Fetch (FC) ⇒ An instruction of 1 or 2 or 3-bytes is extracted from the memory locations during the fetch and stored in the µP's instruction register.

Instruction Execute (EC) ⇒ The instruction is decoded and translated into specific activities during the execution phase. Thus, in an instruction cycle, instruction fetch, and instruction execute cycles are related as depicted in Fig. 2.4. Every instruction cycle consists of 1, 2, 3, 4 or 5-machine cycles as indicated in Fig. 2.5. One machine cycle is required each time the µP access memory or I/O port. The fetch cycle, in general could be 4 to 6-states whereas the execute cycle could of 3 to 6-states. The 1st machine cycle of any instruction is always the fetch cycle that provides identification of the instruction to be executed.

The fetch portion of an instruction cycle requires one machine cycle for each byte of instruction to be fetched. Since instruction is of 1 to 3 bytes long, the instruction fetch is one to 3-machine cycles in duration. The 1st machine cycle in an instruction cycle is always an opcode fetch. The 8-bits obtained during an opcode fetch are always interpreted as the Opcode of an instruction. The machine cycle including wait states is shown in Fig. 2.5



T_1
T_2
T_3
T_4
T_5
T_6
T_1 T_2 T_3 T_4 T_5 T_6 T_1 T_2 T_3 T_1 T_2 T_3 T_1 T_2 T_3 T_1 T_2 T_3
MC_1
Wait
MC_2
MC_3
MC_4
MC_5

Fig.2.5 Machine cycles including wait states

2.11. 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 Figs. 2.6 (a), (b), and (c). The µP 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 µP sends a high on status signal S1 and S0 indicating fetch operation.
- The µP 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.

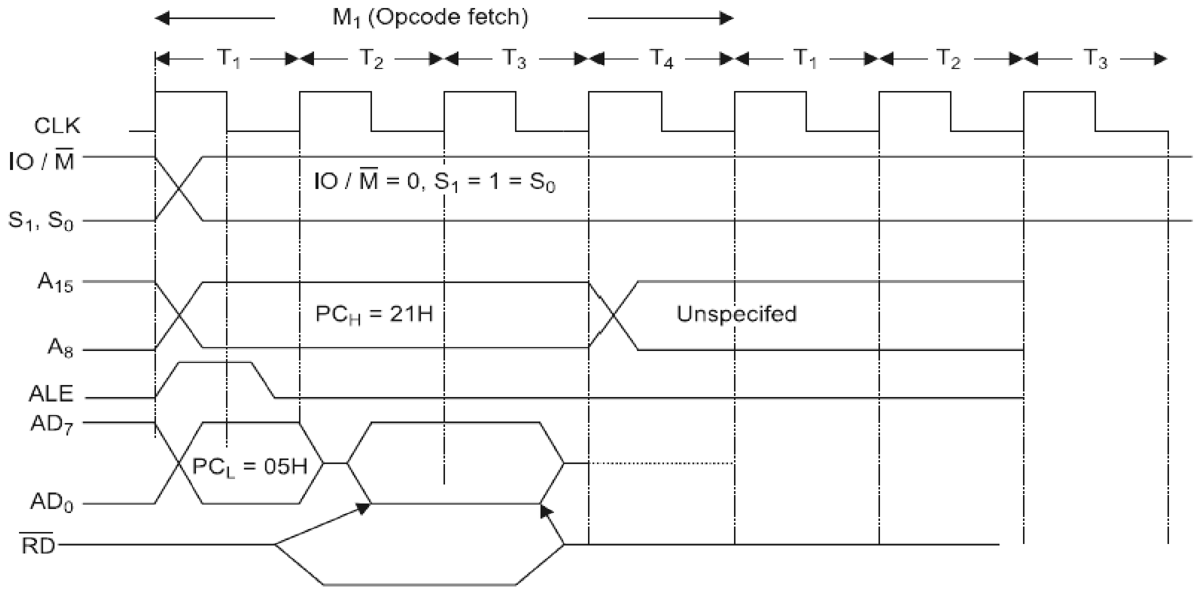


Fig. 2.6(a) Opcode Fetch

• During T3 the RD signal becomes high and memory is disabled.
• During T_4 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 to read/ write data or address from the memory or I/O devices.

Example

Opcode fetch MOV B,C.

T1: The 1st clock of 1st machine cycle (M1) makes ALE high indicating address latch enabled which loads low-order address 00H on AD7⇔AD0 and high-order address 0H simultaneously on A15⇔A8 . The address 00H is latched in T1.

T2: During T2clock, the microprocessor issues RD control signal to enable the memory and memory places 41H from 1000H location on the data bus. Fig. 2.6(b) Data flow from memory to microprocessor

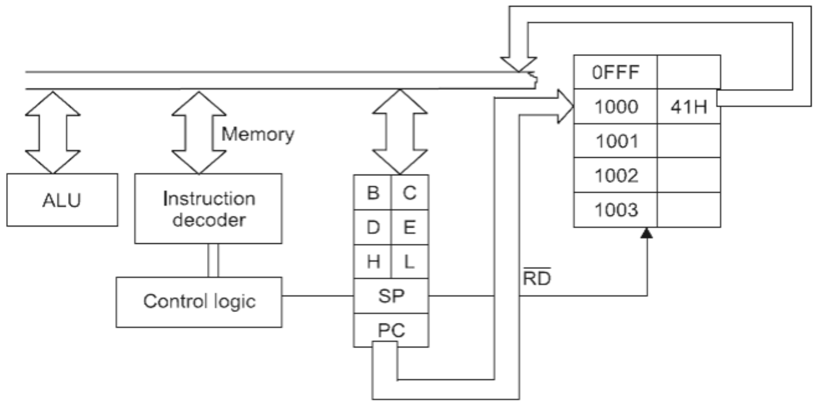


Fig.2.6(b) Data flow from memory to microprocessor

T3: During T3, the 41H is placed in the instruction register and RD = 1 (high) disables signal. It means the memory is disabled in T3 clock cycle. The opcode cycle is completed by end of T3 clock cycle.

T4: The opcode is decoded in T4 clock and the action as per 41H is taken accordingly. In otherword, the content of C-register is copied in B-register. Execution time for opcode 41H is

Clock frequency of 8085 = 3.125 MHz
Time (T) for one clock = 1/3.125 MHz = 325.5 ns = 0.32 µS

Execution time for opcode fetch = 4T = 4*0.32 µS = 1.28 µS

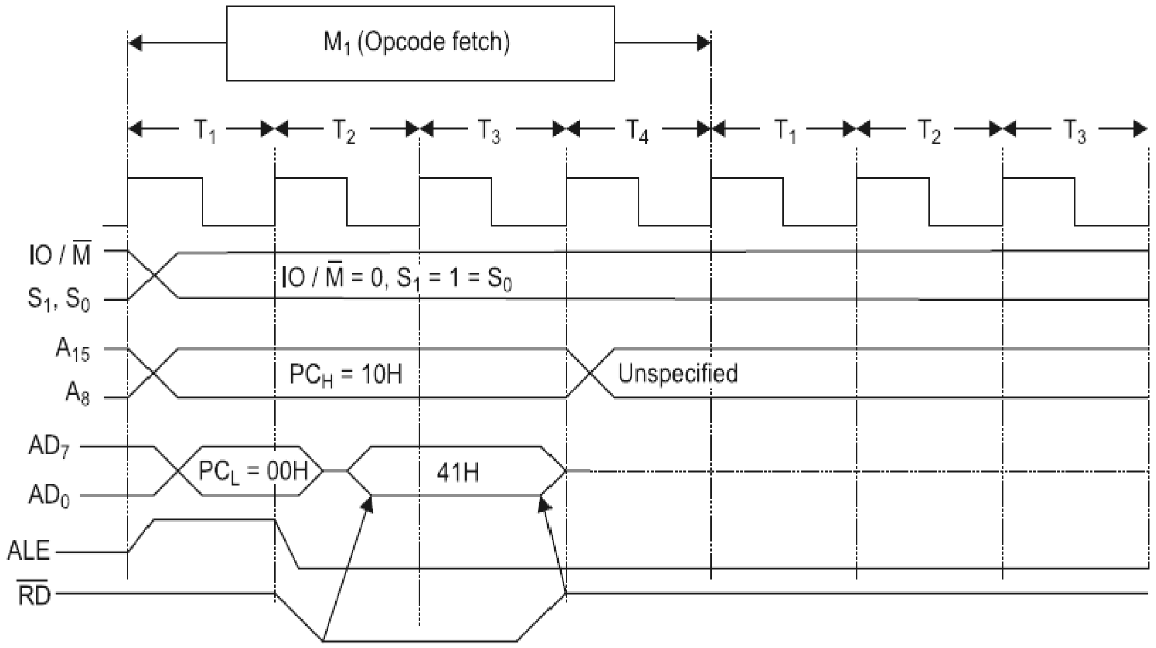


Fig.2.6 (c) Opcode fetch (MOV B,C)

2.12. Read Cycle

The high order address (A15 ⇔A8) and low order address (AD7⇔AD0) are asserted on 1st low going transition of the clock pulse. The timing diagram for IO/ M read are shown in Fig. 2.7 (a) and (b). The A15 ⇔ A8 remains valid in T1, T2, and T3 i.e. duration of the bus cycle, but AD 7 ⇔ AD0 remains valid only in T1. Since it has to remain valid for the whole bus cycle, it must be saved for its use in the T2 and T3. Fig. 2.7 (a) Memory read timing diagram ALE is asserted at the beginning of T1 of each bus cycle and is negated towards the end of T1. ALE is active during T1 only and is used as the clock pulse to latch the address (AD7⇔AD0) during T1. The RD is asserted near the beginning of T2. It ends at the end of T3. As soon as the RD becomes active, it forces the memory or I/O port to assert data. RD becomes inactive towards the end of T3, causing the port or memory to terminate the data.
Fig. 2.7(b) I/O Read timing diagram

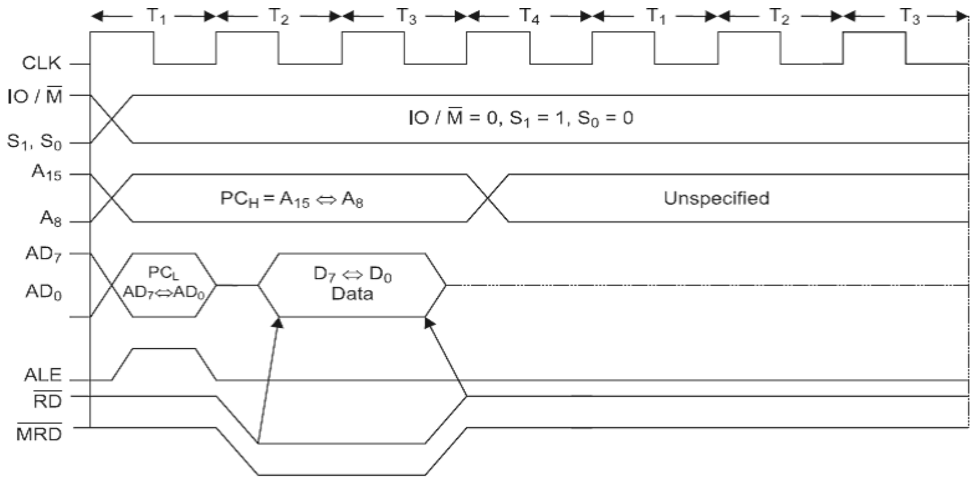


Fig.2.7(a) Memory read timing diagram

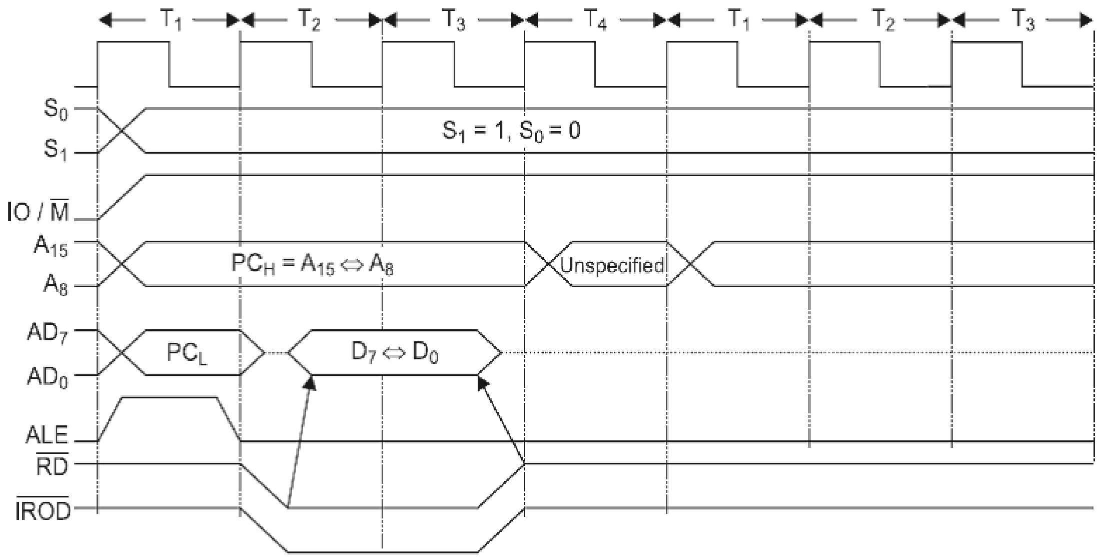


Fig. 2.7 (b) I/O Read Timing diagram

2.13. Write Cycle

Immediately after the termination of the low order address, at the beginning of the T2, data is asserted on the address/data bus by the processor. WR control is activated near the start of T2 and becomes inactive at the end of T3. The processor maintains valid data until after WR is terminated. This ensures that the memory or port has valid data while WR is active. It is clear from Figs. 2.8 (a) and (b) that for READ bus cycle, the data appears on the bus as a result of activating RD and for the WR bus cycle, the time the valid data is on the bus overlaps the time that the WR is active. Fig. 2.8 (a) Memory write timing diagram. Fig. 2.8(b) I/O write timing diagram

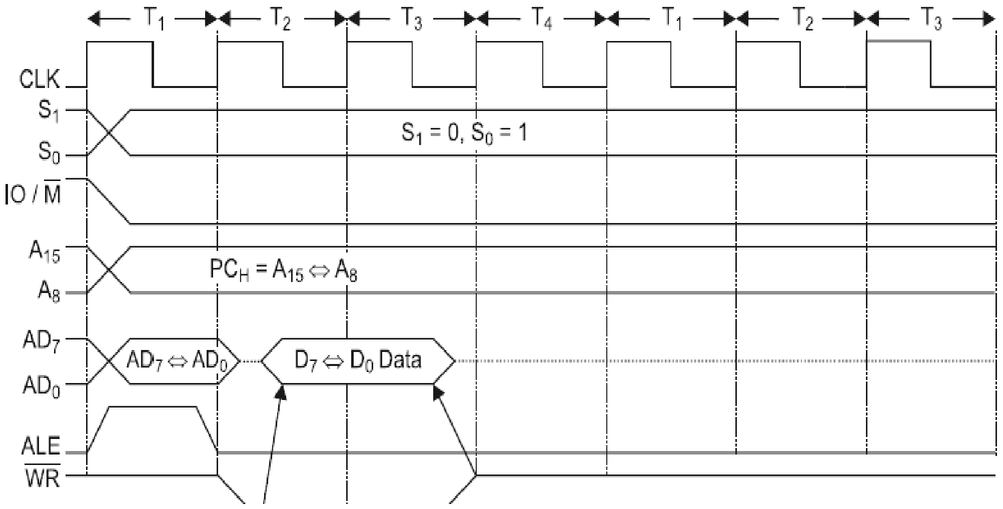


Fig.2.8(a) Memory write timing diagram

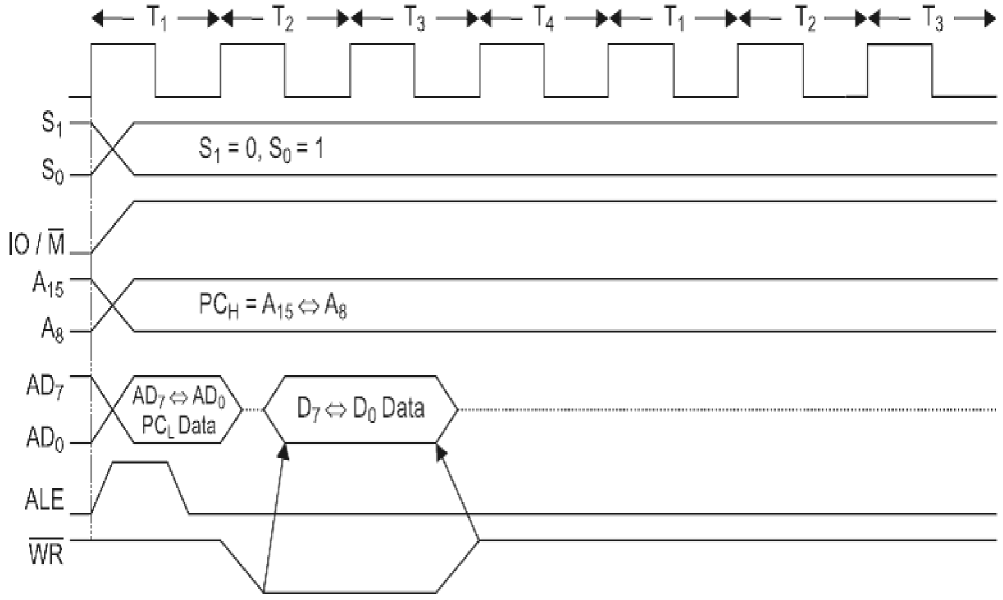


Fig.2.8 (b) I/O write timing diagram

2.14 Programming the 8085A:

A Program is a set of instructions arranged in a sequence to do a specific task.

Programming: It is the process of writing the set of instructions.

Assembly Language Programme: The program with alphanumeric characters or mnemonic is known as assembly language program

Mnemonics: The program makes use of symbolic opcodes known as mnemonics.

An assembly language program (ALP) has the following fields:

1. Label
2. opcode
3. operand
4. comments

The ALP instruction format is shown below

Label	**opcode**	**operand**	**comments**
Loc1	MVI	A, 18H	Move the data 18H to the accumulator

Sample Programs:

1. Write an ALP to perform the addition of two 8-bit numbers (data)

Example: Store the data 24H and 39H in memory locations 1020 and 1021 respectively. Add them and store the result in the memory location 1022.

Memory location	**Mnemonics**	**Opcode**
1000	LDA 1020	3A
1001		20
1002		10
1003	MOV B,A	47
1004	LDA 1021	3A
1005		21
1006		10
1007	ADD B	80
1008	STA 1022	32
1009		22
100A		10
100B	HLT	76

2. Write an ALP to perform the addition of the given two numbers- FFH & 24H

Example: Add the data FFH and 24H and store the result in the memory locations 1050 and 1051.

Memory location	Label	Mnemonics	Opcode
1000		LXI,H 1020	21
1001			20
1002			10
1003		MVI C, 00	0E
1004			00
1005		MOV A,M	7E
1006		INX H	23
1007		ADD M	86
1008		JNC LOOP (100C)	D2
1009			0C
100A			10
100B		INR C	0C
100C	LOOP	STA 1050	32
100 D			50
100E			10
100F		MOV A,C	79
1010		STA 1051	32
1011			51
1012			10
1013		HLT	76

3. Write an ALP to perform the addition of two 16-bit numbers (data)

Example: Add the data F1B2H and 213H and store the result in the memory locations 1090, 1091 and 1092..

Memory location	Label	Mnemonics	Opcode
1000		LXI,H F1B2	21
1001			B2
1002			F1
1003		XCHG	EB
1004		LXI H ,213C	21
1005			3C
1006			21
1007		DAD D	19
1008		SHLD 1092	22
1009			92
100A			10
100B		MVI A, 00	3E
100C			00
100 D		ADC A	8F

100E		STA 1094	32
100F			94
1010			10
1011		HLT	76

4. Write an ALP to perform the subtraction of the given two 8-bit and 16-bit numbers

Example: Subtract two 8-bit numbers and store the result in memory location 2500H

Memory location	Label	Mnemonics	Opcode
1000		LDA 1500	3A
1001			00
1002			15
1003		MOV B, A	47
1004		LDA 1501	3A
1005			01
1006			15
1007		SUB B	90
1008		STA 2500	32
1009			00
100A			25
100B		HLT	76

5. Example: Subtract EA50H from F985H and store the result in memory location 2500 and 2501 using subroutine.

Memory location	Label	Mnemonics	Opcode
1000		MVI B, 00	06
1001			00
1002		LHLD 1050	2A
1003			50
1004			10
1005		XCHG	EB
1006		LHLD 1052	2A
1007			52
1008			10
1009		CALL SUB1	CD
100A			1C
100B			10
100C		DAD D	19
100 D		JC LOOP1	DA
100E			14
100F			10
1010		CALL SUB1	CD

1011			1C
1012			10
1013		INR B	04
1014	LOOP1	SHLD 2500	22
1015			00
1016			25
1017		MOV A,B	78
1018		SHLD 2502	22
1019			02
101A			25
101B		HLT	76
101C	SUB1	MOV A,L	7D
101D		CMA	2F
101E		MOV L,A	6F
101F		MOVA,H	7C
1020		CMA	2F
1021		MOV H,A	67
1022		INX H	23
1023		RET	C9

6. Sum of 8-bit datas in an array

Example: Add a series of 8-bit numbers (5 data) and store the result in memory location

Memory location	Label	Mnemonics	Opcode
1000		LXI H, 1020	21
1001			20
1002			10
1003		MOV C,M	4E
1004		MVI A, 00	00
1005	LOOP	INX H	23
1006		ADD M	86
1007		DCR C	0D
1008		JNZ LOOP	C2
1009		CALL SUB1	CD
100A			06
100B			10
100C		STA 1050	32
100 D			50
100E			10
100F		HLT	76

7. Multiplication of two 8-bit numbers

Example: Multiply two 8-bit numbers stored at memory locations 1050 and 1051 and store the result in memory location 1052.

Memory location	Label	Mnemonics	Opcode
1000		MVI B, 00	06
1001			00
1002		LHLD 1050	2A
1003			50
1004			10
1005		XCHG	EB
1006		LHLD 1052	2A
1007			52
1008			10
1009		CALL SUB1	CD
100A			1C
100B			10
100C		DAD D	19
100 D		JC LOOP1	DA
100E			14
100F			10
1010		CALL SUB1	CD
1011			1C
1012			10
1013		INR B	04
1014	LOOP1	SHLD 2500	22
1015			00
1016			25
1017		MOV A,B	78
1018		SHLD 2502	22
1019			02
101A			25
101B		HLT	76
101C	SUB1	MOV A,L	7D
101D		CMA	2F
101E		MOV L,A	6F
101F		MOVA,H	7C
1020		CMA	2F
1021		MOV H,A	67
1022		INX H	23
1023		RET	C9

8. Division of two 8-bit numbers

Example: Write an ALP to divide two 8-bit numbers

Label	Mnemonics	Comment
	LDA 6501 H	Load the divisor in accumulator
	MOV B, A	Move the divisor to B register
	LDA 6500 H	Load the dividend in accumulator
	MVI C, 00H	Clear C register to store quotient
STEP 1:	CMP B	Compare the content of A and B
	JC STEP 2	If divisor < dividend , go to STEP 2
	SUB B	Subtract divisor from dividend
	INR C	Increment quotient
	JMP STEP 1	continue the subtraction
STEP 2	STA 6503H	Store the accumulator
	MOV A, C	Move the content of C to A
	STA 6502 H	Store the quotient

9. Ascending order – Sorting of Numbers

Example: Write an ALP to arrange the data bytes in acending order (sorting of numbers)

Memory Address	Label	Mnemonics	Comment
4100H		LDA 4300 H	Load the number of passes from memory into acc.
4103 H		MOV B, A	Move the data from the acc. to register B
4104 H	LOC 5:	MOV C, B	Move the data from register B into the register C
4105 H		LXI H 4400 H	Load H-L pair with memory address
4108 H	LOC 3:	MOV A, M	Move the data from memory location to acc.
4109 H		INX H	Increment the content of H-L pair
410A H		CMP M	Compare the content of M with acc.
410B H		JC LOC 1	If carry = 1, Go to LOC 1.
410E H		MOV D, M	The data in memory pointed by H-L pair is transferred to register D
4110 H		DCX H	Decrement the content of H-L pair
4111 H		MOV M, D	Move the data in register D to H-L pair
4112 H		INX H	Increment the content of the H-L pair
4113 H	LOC 1:	DCR C	Decrement the register C
4114 H		JZ LOC 2	If the counter ≠ 0, jump to LOC 3
4117 H		JMP LOC 3	If the counter = 0, jump to LOC 2
411A H	LOC 2:	DCR B	Decrement register B (pass counter)
411B H		JZ LOC 4	If the counter = 0, jump to LOC 4
411E H		JMP LOC 5	If the counter ≠ 0, jump to LOC 5

4121 H	LOC 4:	HLT	Stop

10. To find Two's compliment

Example: Write an ALP to find Two's compliment of a 16-bit number

Label	**Mnemonics**	**Comment**
	LXI H, 6501 H	Address of 8 LSBs of the given no.
	MOV B, 00 H	Clear B register
	MOV A, M	Move 8 LSBs to accumulator
	CMA	1's compliment of 8 LSBs
	ADI 01 H	2's compliment of 8 LSBs
	STA 6503 H	Store 8 LSBs of the result
	JNC STEP	Jump on no carry to "STEP"
	INR B	If carry is available, store in B
STEP:	INX H	Address of 8 MSBs of the given no.
	MOV A, M	Move 8 MSBs to accumulator
	CMA	1's compliment
	ADD B	Add carry
	STA 6504 H	Store 8 MSBs of the result
	HLT	Halt

Unit – III

Peripheral Interfacing Devices and Techniques

3.1 What is meant by memory address space?

It is the maximum possible memory size which can be used with µp.

(i) Address space partitioning:

The allocation of address to memory chips and I/O devices depends on the µp architecture. Some processors provide only one address space thereby treating I/O devices as memory locations. Intel 8085 uses a 16- bit wide address bus for addressing memories and I/O devices, using 16- bit wide address bus it can access $2^{16} = 64k$ bytes of memory and I/O devices. The 64k address are to be assigned to memories and I/O devices for their addressing. There are two schemes for the allocation of address to memories and input/output devices.

(a) Memory mapped I/O scheme

(b) I/O mapped (standard I/O) I/O scheme

(a) Memory mapped I/O scheme:

I/O devices are connected with the place memory address space called memory mapped I/O scheme. In this type of I/O, the MPU uses 16 address lines to identity an I/O device, an I/O is connected as if it is memory resisters. This is known as memory mapped I/O. The MPU uses the same control signal(memory read or memory write) and instructions as those of memory.

To transfer data between the MPU and I/O devices, memory related instructions (such as LDA, STA etc) and memory control signal ($\overline{MEMR}$ and $\overline{MEMW}$) are used. The microprocessor communicates with an I/O device as if it were one of the memory locations.

(b) I/O mapped I/O scheme:

When separate address space is available for I/O device, the I/O devices are connected in this space. This is I/O mapped scheme. In this scheme, the address assigned to memory locations can also be assigned to I/O devices. Since the same address may be assigned to a memory location or an I/O device; The µp must issue a signal to distinguish whether the address on the address bus if for a memory location or I/O device. The Intel 8085 issues an IO/$\overline{M}$ signal for this purpose. When this signal is high the address bus is for an I/O device. When the signal is low, the address on the address bus is for a memory location. Two extra instructions IN/OUT are used to address I/O devices. The IN instructions is used to lead the data of an input device. The output instruction is used to send data to an output device. This scheme is suitable for a large system.

Typically, to display the contents of the accumulator at an output devices.[such as LEDs] with the address for example, 01H, the instructions will be written and stored in memory as follows:

Memory address	Machine codes	Mnemonics
2050	D_3	Out 01 H
2051	01H	

3.2 Difference between memory mapped I/O and I/O mapped I/O scheme:

Memory mapped I/O	I/O mapped I/O
I/O ports are also considered as memory locations.	These are not considered as memory locations
Considerable memory location are allotted for I/O ports where instruction and data cannot be stored	No such problem.
The I/O ports have 16 bit address. Therefore 64k is shared between I/O system and memory system	The port address will be only 8 bit. Therefore 256 separate I/O devices can be connected. The I/O map is independent of memory map.
Data transfer between any CPU register and I/O ports are possible.	Data transfer is possible only in A register and I/O ports.
LDA,STA instruction are used for I/O operation and also MOV M, R, MOV R,M instructions can be used. So I/O operations are not self-evident from the program listing.	Only IN, OUT instruction are used. So I/O operation are clearly evident from program listing.
Main memory space is reduced.	Separate memory is used for I/O ports instructions
Program debugging is difficult	Program debugging is easy
Decoding the I/O address is difficult.	Simple I/O address system. So decoding is easier.
Control signal used for Input output are $\overline{MR}/\overline{MW}$	Control signals are used for input/output are $\overline{IDR}/\overline{IOW}$
Arithmetic or logical operations can be directly performed with I/O data.	Not possible

3.3 What is meant by I/O address space?

It is the maximum possible number of I/O device which can be interfaced with a µp.

I/O map: The entire range of I/O addresses from 00 to FF is known as I/O map.

Interfacing: A microprocessor combine with memory and input/output devices forms a microcomputer. The µp is the heart of a computer. Mnemonics and Input / Output devices are interfaced to µp to form a microcomputer. In case of large and mini computers the memories and input /output devices are interfaced to CPU by the manufactures. In a µp- based system the designs has to select suitable memories and I/O devices for his task and interface them to the µp. The selected memories and I/O devices should be compatible, additional electronic circuit has to be designed through which the device may be interfaced to the CPU. Fig 3.1. Shows a schematic diagram to interface memory chips or I/O devices to a µp . An address decoding circuit is employed to select the required I/O device or a memory chip.

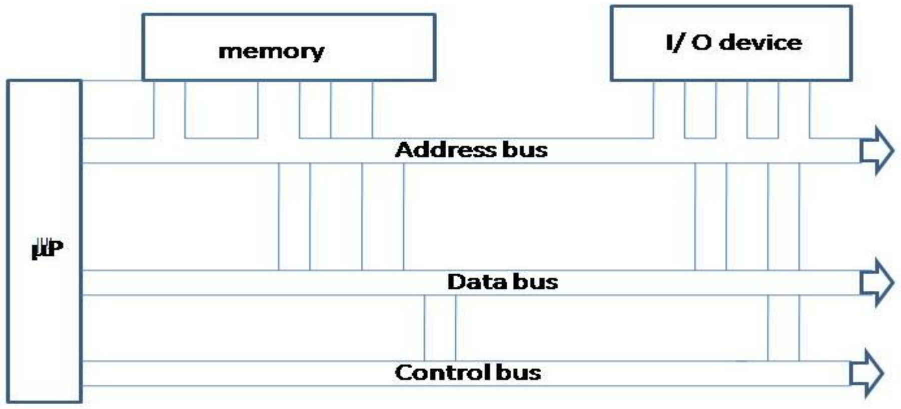


Fig.3.1

Figure 3.2 shows a schematic diagram of a decoding circuit. If IO/$\overline{M}$ is high the detector 2 is activated and the required I/O device is selected. If is IO/$\overline{M}$ low, the decoder 1 is activated and the required memory chip is selected. A flow MSBs of the address lines are applied to the decodes to select a memory chip or an I/O device. The function of incoding is 10 generate a binary code, and the process to generating codes is known as incoding . The decoding is the reverse process of incoding BCD hexa OCTAL, ASCII, EBCDIC etc.

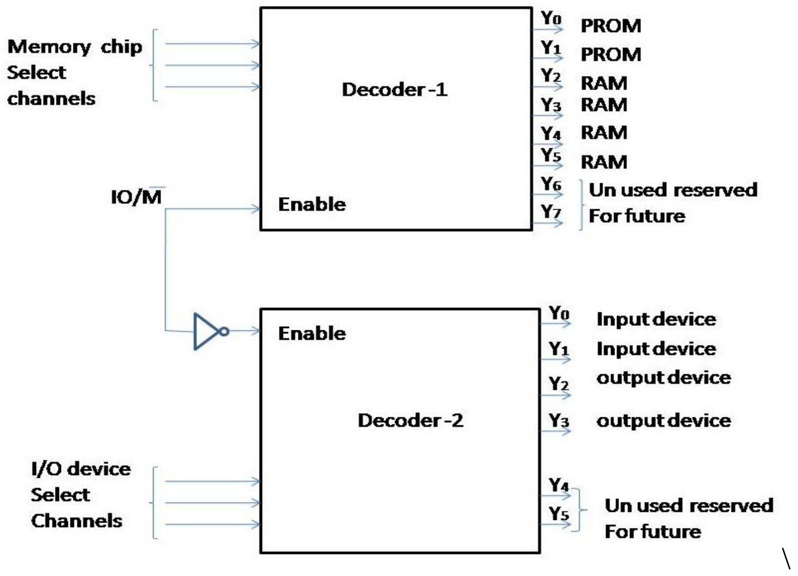


Fig.3.2

3.4 Memory Interfacing:

The address of a memory location or an I/O device is sent out by the µp. The corresponding memory chip or I/O device is selected by a decoding circuit. The decoding task can be performed by a decoder, a comparator, a bipolar PROM or PLA (Programmed Logic Array). The most commonly used decodes is 74LS138. If is a 1 to 8 times decoder. Figure 3.3 shows the interface of memory chips through 74L3S138. G_1,G_2A and G_2B are enable signals. To enable 74LS138, G_1 should be high and G_2A and G_2B should be low. A, B and C are select lines. By applying proper logic to select lines any one the outputs can be selected. $y_0, y_1, \dots \dots \dots, y_7$ are 8 outputs lines. An output line goes low when it is selected other output lines remain high. Table 3.1 shows the truth table for 74LS138, when G_1 is low or G_2A is high or G_2B is high, all outputs lines become high. Thus 74LS138 acts as decoder only when G_1 is high and G_2A and G_2B are low.

Input						Output							
Enable			Select										
G_1	G_2A	G_2B	C	B	A	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
X	H	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	L	H	H	H	H	H	L	H	H	H	H
H	L	L	H	L	L	H	H	H	H	L	H	H	H
H	L	L	H	L	H	H	H	H	H	H	L	H	H
H	L	L	H	H	L	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	l

Table 3.1

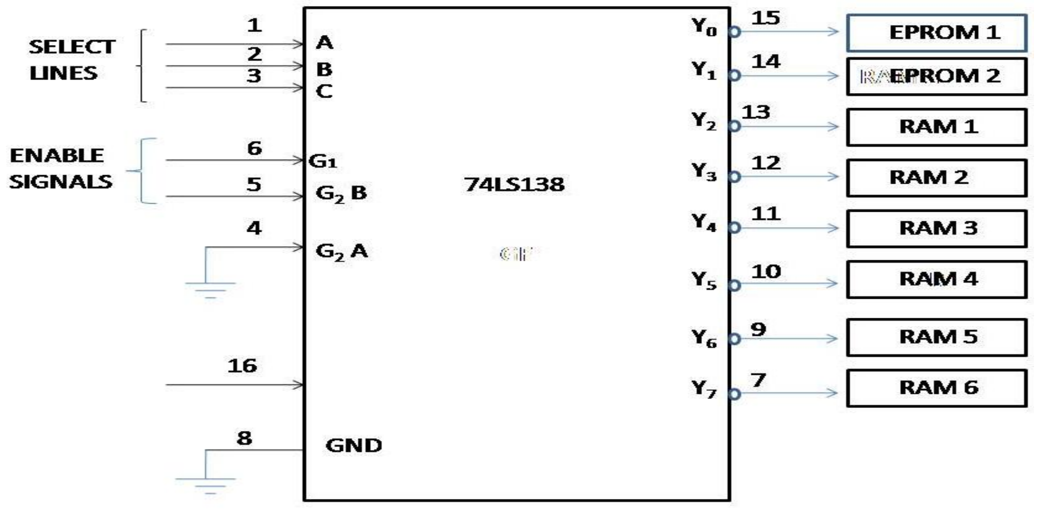


Fig 3.3. Interfacing of memory chips using 74LS138

The memory locations for EPROM1 will lie in the range 0000 to 1FFF. These are the memory locations for zone 0 for the memory chip which is connected to the output line Y_0 of the decodes. Similarly for zone 1 is 2000 to 3FFF and zone 7 is E000 to FFFF. Table 2 shows the memory locations for various zones.

Decodes output	Memory divide	Zone of the address space	Memory locations address
Y_0	EPROM1	Zone 0	0000 to 1FFF
Y_1	EPROM2	Zone1	2000 to 3FFF
Y_2	RAM1	Zone2	4000 to 5FFF
Y_3	RAM2	Zone3	6000 to 7FFF
Y_4	RAM3	Zone4	8000 to 9FFF
Y_5	RAM4	Zone5	A000 to BFFF
Y_6	RAM5	Zone 6	C000 to DFFF
Y_7	Ram6	Zone7	E000 to FFFF

Table 3.2: memory locations for various zone

The entire memory address (64k for 8085) has been divided into 8 zones Address lines A_{15} , A_{14} and A_{13} have been applied to the select lines A, B and C of the 74LSI38. The logic applied to these lines selects a particular memory device, an EPROM or a RAM other address lines A_0 , A_1 ,………, A_{12} go directly to the memory chip. They decode the address of the memory location within a selected memory chip. IO/$\overline{M}$ is connected to G_2B , where IO/$\overline{M}$ goes low for memory read/write operation; G_2B goes low, G_1 is connected to +5V Vdc; supply and G_2A is grounded.

3.5 I/O interfacing:

Figure 3.4 shows the interface of I/O devices through decodes 74LS138. As the address bus are used for I/O addressing. The address lines A_8, A_9 and A_{10} have been applied to select lines A,

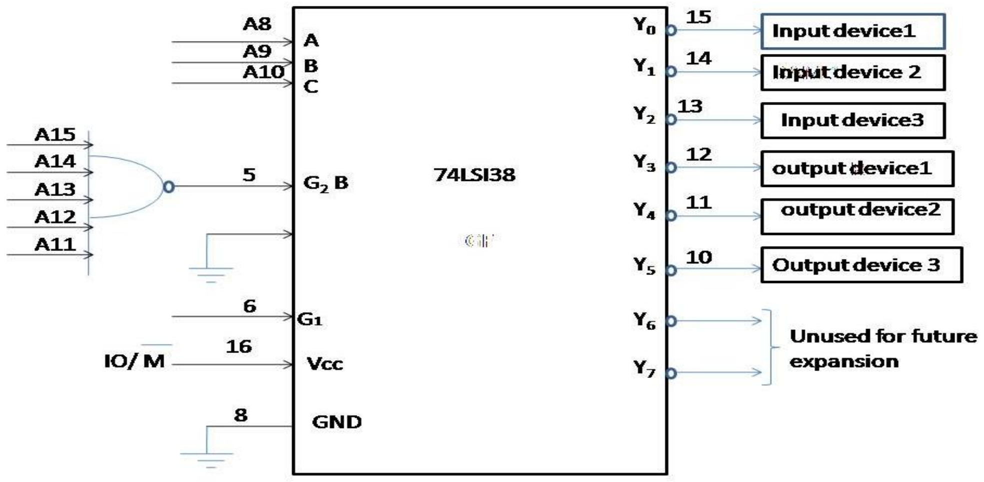


Fig 3.4: Interfacing of I/O devices using 74LS138.

B and C of the 74LS138. The address lines A_{11}- A_{15} are applied G_2B through a NAND Gate. G_2B becomes low only when all address lines A_{11} – A_{15} are high. G_2A is grounded. IO/$\overline{M}$ is connected to G_1. When IO/$\overline{M}$ goes high for I/O read/ write operation. G_1 goes high. Table 3.3 shows the address of I/O devices to 741S138.

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	Selected output lines	Corresponding Address	I/O device
1	1	1	1	1	0	0	0	Y_0	F_8	Input device1
1	1	1	1	1	0	0	1	Y_1	F_9	Input device2
1	1	1	1	1	0	1	0	Y_2	F_A	Input device3
1	1	1	1	1	0	1	1	Y_3	F_B	Output device1
1	1	1	1	1	1	0	0	Y_4	F_C	Output device2
1	1	1	1	1	1	0	1	Y_5	F_D	Output device3
1	1	1	1	1	1	1	0	Y_6	F_E	Unused
1	1	1	1	1	1	1	1	Y_7	F_F	Unused

Table 3.3: Address of I/O devices connected to 74LS138

3.6 Data transfer scheme:

To solve the problem of mismatch between a µp and I/O devices a number of data transfer techniques have been developed. The data transferred between µp and I/O devices is transferred by one of the following two methods.

(a) Programmed data transfers.

(b) Direct memory access transfer.

(a) Programmed data transfers:

Program data transfer scheme are controlled by CPU. Data transfer between µp and I/O devices an under the control of programs which reside in the memory. These programs are executed by the CPU where an I/O device is ready to transfer data. Programmed data transfer are generally used when a relatively small amount of data is transferred with relatively with relatively slow I/O devices. Eg: AD, D/A converters etc. In these schemes usually one byte of data is to be transferred at a time. Programmed data transfer can be further classified as,

(i) Synchronous data transfer.
(ii) Asynchronous data transfer.
(iii) Interrupt driven transfer.

(i) Synchronous data transfer:

Synchronous means "at the same time". When the I/O device and µp match in speed, the synchronous transfer method is used. Whenever data is to be obtained from the device or transferred to the device, the user program may issue suitable instruction addressing the device. At the end of the execution of this instruction, the transfer would have been completed. The I/O devices compatible with µp in speed are usually not available. Hence the techniques of data transfer is rarely used for I/O devices. However memory compatible with µp are available, and therefore this technique is invariably used with compatible memory devices. This is the simplest of all data transfer schemes. If an output device connected to the 8085in memory mapped mode, the following instruction may be used for transferring ACC counter to the device.

MOV m, A

assuming that the address of the device is already stored in HL pair. If the device is connected in I/O mapped mode, then the out instruction may be issued

OUT 2

assuming that 2 is the port number associated with this device. Similarly for an input device, the instruction

MOV A,M or IN 2.

There are two data transfer synchronizing techniques are available. They are polling and Interrupts.

(ii) Asynchronous data transfers:

Asynchronous means "at irregular intervals". When the I/O device and µp speeds do not match, asynchronous data transfer may be used. According to this scheme, the µp issues first get ready instruction to the devices, subsequently the µp keeps waiting until the device get ready. The µp processes issues a data transfer instruction immediately after the devices get ready. In order to work, the device should provide a signal which may be tested by the µp to as certain whether it is ready or not. This form of data transfer is also known as Hand shaking; since some signals are exchanged between the I/O device and the µp before the actual data transfer takes place. The µp issues an initiating signal to the I/O device to get ready (or to start). Where I/O device becomes ready it sends signals to the processor to indicate that it is ready. Such signals are called hand shake signals. Fig 3.5 shows a schematic diagram for asynchronous data transfer.

Asynchronous data transfer is used for slow I/O device. This techniques is an inefficient technique because the precious time of the µp is wasted in waiting.

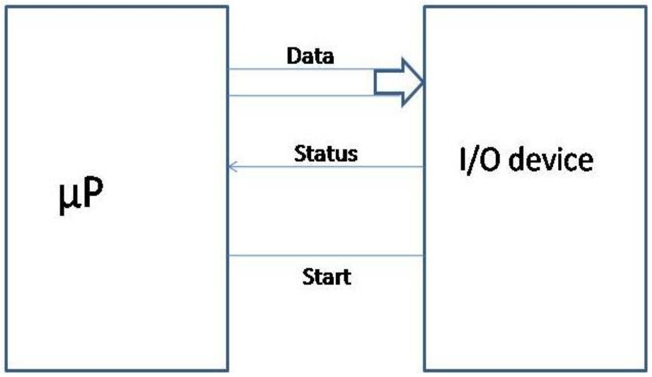


Fig.3.5

It may be illustrated as follows.

Begin: Issue instruction to device to get ready.

Repeat: test device ready flag;

Until device ready;

Issue instruction to transfer data.

End: The hand shaking may be either software or hardware.

(iii) Interrupt driven data transfer:

It is an efficient technique as compared with the asynchronous data transfer because precious time of the µp is not wasted in waiting while an I/O device is getting ready. In this scheme the µp initiates an I/O device to get ready and then it executes its main program instead of remaining in a program loop to check the status of the I/O device. When the I/O device becomes ready to transfer data, it sends a high signal to the µp through a special input line called an interrupt line. In other wards it interrupts the normal processing sequence of the µp. On receiving an interrupt, the µp completes the current instruction hand and then attends the I/O devices. It saves the contents of the program counter on the stack first, and then takes up a subroutine called ISS(Interrupt Service Subroutine). In executes ISS to transfer data from or to the ISS device. Different ISS are to be provided for different I/O devices. After completing the data transfer, the µp returns back to the main program which it was executing before the interrupt was occurred. Interrupt driven data transfer is used for slow I/O devices.

Example: The following Fig.3.6 shows the interfacing of an A/D converter to transfer data employing interrupt driven data transfer scheme. The µp sends first, the start a conversion signal, S/C to the A/D converter. Therefore the µp executes its main program. A/D converter is a slow device compared to the µp. It takes some time to convert analog signal to its equivalent digital quantity. When A.D converter completes the task of conversion, it makes an end of conversion signal, E/C high,. The E/C signal is connected to an interrupt line of the µp. when interrupt line goes high, the µp takes all necessary steps to transfer data from the A/D converter. After completing the data transfer, the µp returns back to execute the main program that it was executing prior to the interrupt.

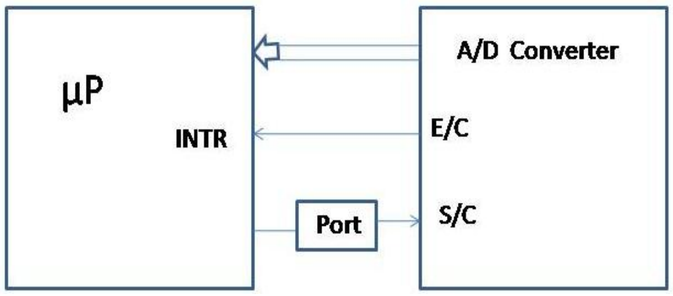


Fig.3.6 Interrupt Driven data transfer scheme for an A/D converter.

3.7 DMA data transfer scheme:

In DMA data transfer scheme CPU does not participate. Data are directly transferred from an I/O device to the memory device or vice versa. The data transfer is controlled by the I/O device or a DMA controller. When a large block of data is to be transferred, DMA is used. It bulk of data are transferred through CPU, it takes appreciable time and process becomes slow. An I/O devices which wants to send data using DMA technique, sends to HOLD signal from an I/O device, the CPU gives the control of buses as soon as the current machine cycle is completed. The CPU sends HOLD acknowledgement signal to the I/O device to indicate that it has received the HOLD request and it will give up the buses in the next machine cycle. The I/O devices takes over the control of buses and directly transfer data to the memory ar reads data from the memory. DMA transfer scheme is a faster scheme as compared to programmed data transfer scheme. It is used to transfer data from mass storage devices such as hand disks, floppy disks etc, it also used for high-speed pointers.

DMA data transfer schemes are of the following three types.

i. Burst mode(or) Visible DMA data transfer
ii. Cycle stealing techniques (or) Transparent of DMA transfer scheme.
iii. Demand transfer mode DMA.

(i) Burst mode or visible DMA data transfer:

A scheme of DMA data transfer in which the I/O device withdraws the DMA request only after all the data bytes have been transferred is called burst mode data transfer. By this technique a block of data is transferred. This technique is employed by magnetic disk drives. In the middle, magnetic disks transfer cannot be stopped or slowed down without loss of data. hence block transfer of data is a must cycle stealing techniques or Transparent DMA transfer schemes. In this technique a long block of data is transferred by a sequence of DMA cycles. In this method, after transferring one byte or several bytes, the I/O device withdrawn the DMA request. This method reduces the interference in CPUs activities. The interference can be eliminated completely by designing an interfacing circuitry which can steel but cycle only when the CPU is not using the system bus.

(ii) Demand transfer mode DMA:

Whenever the demand is created by the I/O device or memory device, the bulk of data transfer taking place. In DMA data transfer schemes I/O devices control or data transfer and hence the I/O devices must have register to store memory addresses and byte count. It must also have some electronics circuitry to generate necessary control signals required during DMA operations. Usually I/O devices do not have these facilities. To solve these problems DMA controllers have been designed and developed. Examples of DMA controller have been designed and developed. Examples of DMA controllers chips are Intel 8237A, 8257 etc. In Fig.3.7 shows some various types of data transfer schemes.

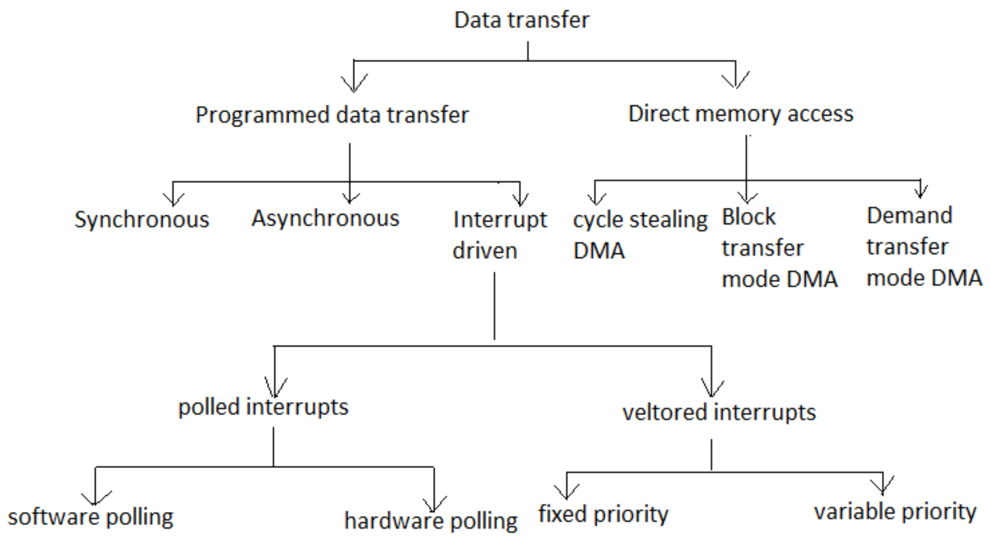


Fig.3.7

3.8: 8085 interrupt system:

Interrupt is a signal send by an external device to the processor so as to request the processer to program a particular task or work.

They are three methods of classifying interrupts.

Method 1: The interrupts are classified into hardware and software interrupts.

Method 2: The interrupts are classified into vectored and non-vectored interrupts.

Method 3: The interrupts are classified into Maskable and Nonmaskable interrupts.

(i) Software interrupts of 8085A:

The software interrupts are program instructions when the instruction is executed, the processor executes an interrupts service routine(ISR) stored in the vector address of the software interrupt instruction. The software interrupts of 8085 are RSTO, RST1, RST2, RST3, RST4, RST5, RST6 and RST7. The vector address of software interrupts are given below:

Interrupts	Vector address
RST0	0000_H
RST1	0008_H
RST2	0010_H
RST3	0018_H
RST4	0020_H
RST5	0028_H
RST6	0030_H
RST7	0038_H

When the microprocessor recognizes an interrupt, it saves the processor status in stack. Then it call and execute an Interrupt Service Routine (ISR). At the end of ISR it restores the processor status and the program control is transferred to main program. In this way a microprocessor service an interrupt request.

The software interrupts of 8085 are vectored interrupts. The software interrupts can not be masked and they cannot be disabled.

(ii) Hardware interrupts of 8085A:

The hardware interrupts of 8085 are initiated by an external device by planning an appropriate signal at the interrupt pin of the processor. The processor keeps on checking the interrupts pin at the second T-state of last machine cycle of every instruction. If the processor finds a valid interrupts signal and if the interrupt is unmasked and enabled, then the processor accepts the interrupts. The acceptance of the interrupts is acknowledged by sending an $\overline{INTR}$ signal to the interrupted device.

The hardware interrupts of 8085 are TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR. The TRAP, RST 7.5, RST6.5 and RST 5.5 are vectored interrupts. In vectored interrupts the address to which the program control is transfer(when the interrupt is accepted) is fixed by the manufacturer. The vector addresses of hardware interrupts are given below:

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

INTR and its expansion:

The INTR is general interrupt request. An external device can interrupt the processor by placing a high signal on INTR pin of 8085. If the processor accepts the interrupt ,then it will send an acknowledge signal $\overline{INTR}$ to the interrupted device. On receiving the acknowledge signal the interrupted device has to place either an RST_n code and the data bus. On receiving the RST_n opcode the 8085 processor generate the vector address of RST_n instruction. It saves the content of program of RST_n instruction. It saves the content of program counter (pc) in stack. Then it loads the vector address in PC and executes an Interrupt Service Routine (ISR) stored at this address.

(iii) Vectored and non-vectored interrupt:

Vectoring is the process of generating the address of interrupt service routine to be loaded in the program counter. When an interrupt is accepted, if the processes control branches to a specified address defined by the manufacturer then the interrupts is called vectored interrupts. In Non-vectored interrupt, there is no specific address for storing the interrupt service routine. Hence the interrupted device should give the address of the ISR.

3.9 Interfacing devices and I/O devices:

To communicate with the outside world, µcs use peripherals (I/O devices). Commonly used peripherals are: A/D converter, D/A converter, CRT, Printers, hard disk, floppy disks, magnetic tapes etc. Peripherals are connected to the µc through electronic circuits known as interfacing circuits. Generally each I/O device requires a separate interfacing circuit. The interfacing circuit converts the data available from an input device into compatible format the computer. The interface associated with the output device converts the output of the µp into the desired peripheral format, some of the general purpose interfacing devices are,

1. I/O port
2. Programmable peripheral interface(PPI)
3. DMA controller
4. Interrupt controller
5. Communication interface
6. Programmable counter/ Internal timer

Special purpose interfacing devices are designed to interface a particular type of I/O device to the μp. Example of such devices are:

1. CRT controller
2. Floppy disk controller
3. Keyboard and display interface.

3.10 Generation of control signals for memory and I/O devices:

Intel 8085 issues control signals $\overline{RD}, \overline{WR}$ for read and write operation of memory and I/O devices. It also issues a status signal $IO/\overline{M}$ to distinguish whether read/write operation is to be performed by memory or I/O device. Memory and I/O devices require control signals is modified from shown below.

$\overline{MEMR}$→memory read

$\overline{MEMW}$→memory write

$\overline{IOR}$→ I/O read

$\overline{IOW}$→ I/O write

These control signals are generated using $\overline{RD}$, $\overline{WR}$ and $IO/\overline{M}$ using logic gates. OR logic and inverters are used for the purpose as shown in figure.

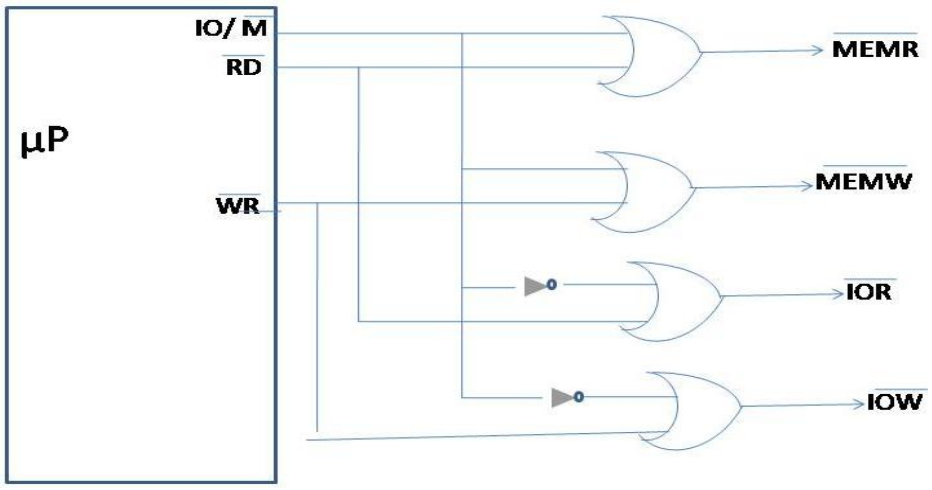


Figure 3.8 : control signals for memory I/O Read/Write operation.

To get $\overline{MEMR}$, use $IO/\overline{M}$ V $\overline{RD}$

Here V is a symbol for logical OR operations. Memory read operation takes place when $IO/\overline{M}$ and $\overline{RD}$ both are low. $IO/\overline{M}$ and $\overline{RD}$ are applied to an OR gate. The output of the OR gate is

$\overline{MEMR}$ when both . $IO/\overline{M}$ and $\overline{RD}$ are low. $\overline{MEMR}$ goes low and it activates memory for read operation. Similarly, other control signals are obtained as shown below:

To get $\overline{MEMW}$, use $IO/\overline{M}$ V $\overline{WR}$ I/O read operation takes place when $IO/\overline{M}$ is high. To get $\overline{\text{IOR}}$ and $\overline{IOW}$ signals $IO/\overline{M}$ is inverted and then applied to OR gates.

Get $\overline{IOR}$ using inverted $IO/\overline{M}$ V $\overline{RD}$

Get $\overline{IOW}$ using inverted $IO/\overline{M}$ V $\overline{WR}$

3.11 I/O ports:

An input device is connected to the µp through an input port. An input port is a place for unloading data. An input device unloads data into port. The µp reads data from the input port. Thus datas are transferred from the input device to the accumulator is a input port. Similarly an output device is connected to the µp through an output port. The µp unloads data into an output port. As the output port is connected to the output device. Data are transferred to the output device. Figure shows a schematic connection of the CPU I/O ports and I/O devices.

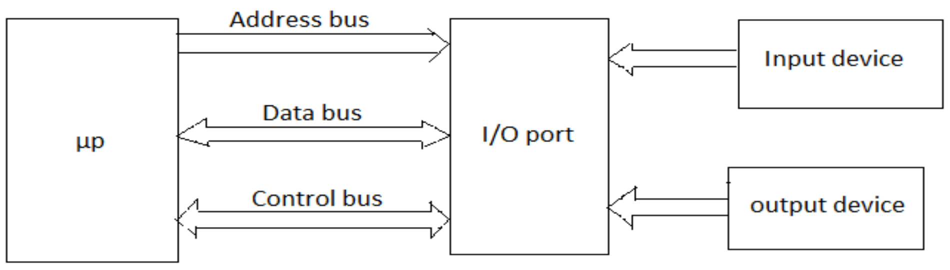
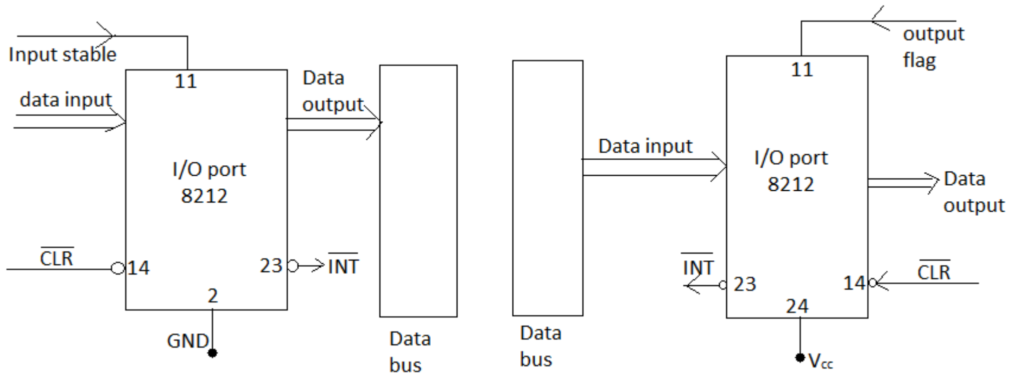


Figure 3.9 : Interfacing of I/O device through I/O port

An I/O port may be programmable or non-programmable. A non-programmable port behaves as an input port of it has been designed and connected in input mode. Similarly a port connected in an output acts as an output port. But a programmable I/O port can be programmed to act either as an input port or output port, the electrical connections remain some.

The Intel 8212 is an 8- bit non-programmable I/O port. It can be connected to the µp either as an input port or an output port. If we require one input port and one output port, two units of 8212 will be required care of them will be connecting input mode and the others in the output mode. Figure 3.10 shows the connections of 8212 in input mode and output mode. The SCL 6532 is a RAM, I/O , Internal times device(RIOT) manufactured by semiconductors complex ltd(India). It has a 8- bit, bidirectional data bus; 128 x 8 state RAM, two 8- bit bidirectional data ports; programmable internal times with interrupt capability 6502/6800 bus compabability. It operates with 1 MHz and 2 MHz clock and a single supply +5v. It is implemented in a 40-pin IC.

The Intel 8155 is a RAM with I/O ports. It contains a 256 byte RAM, 3 I/O ports and a 14-bit timer/counter. There are 3 ports: A, B and C. The port A and port B are 8 bit and port c of 6 bits. Each port can be programmed either as an input port or output port. The port c may also be programmed as a control port for the port A and port B.



(a) Input mode (b) output mode

Fig.3.10

3.12 Programmable Peripheral Interface (PPI):

A programmable peripheral interface is a multiport device. The ports may be programmed in a variety of ways as required by the programmer. The device is very useful for interfacing peripheral device. The term PIA, Peripheral Interface Adapter is also used by some manufacturer, Indian kits generally use Intel 8255 which is a programmable peripheral interface (PPI).

Intel 8255:

The Intel 8255 is a programmable peripheral interface (PPI). It has two versions namely, the Intel 8255A and the Intel 8255A-5 . General discriptions for both are same. There are same differences in their electrical characteristics. It's main functions are to interface peripheral device to the µp. It has 3, 8-bit ports, namely port A, port B and port C. The port C has been further divided into two 4-bits ports port C upper and port C lower. Thus a total of 4 ports. Each port can be programmed either as an input port or as an output port.

Architecture of Intel 8255A:

Figure 3.11 shows the pin diagram of Intel 8255A. It is a 40 pin I.C package. It operates on a single 5Vd.c supply. Its important characteristics are as follows

Ambient temperature 0 to 70^{0}c.

Voltage on any pin 0.5v to 7v.

Power dissipation 1 watt.

V_{IL}= Input low voltage =minimum 0.5V, maximum 0.8V.

V_{IH}= Input high voltage= minimum 2V, maximum V_{CC}.

V_{OL}=output low voltage=0.45 V

V_{OH}= output high voltage=2.4V

I_{DR}= Darkington drive current minimum 1 mA, maximum 4 mA on any 8 pins of port.

The pins for various ports are as follows:

$P_{A0} - P_{A7}$	8 pins of port A
$P_{B0} - P_{B7}$	8 pins of port B
$P_{C0} - P_{C3}$	4 pins of port C lower
$P_{C4} - P_{C7}$	4 pins of port C upper

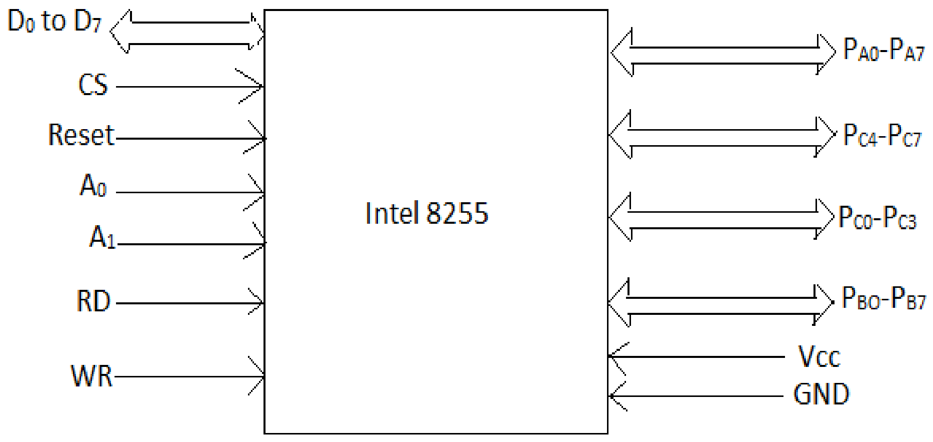


Fig.3.11 The pin diagram of Intel 8255A

The important control signals are as follows:

<u>$\overline{CS}$ (chip select):</u>

It is a chip select signal. The low status of this signal enables communication between the CPU and 8255.

<u>$\overline{RD}$ (read):</u>

When $\overline{RD}$ goes low, the 8255 sends out data or status information to the CPU on the data bus. In other words it allows the CPU to read data from the input port of 8255.

$\overline{WR}$ (write):

When $\overline{WR}$ goes low, the CPU writes data or control word into 8255. The CPU writes data into the output port of 8255 and the control word into the control word register.

A_0 *and* A_1:

The selection of input port and control word register is done using A_0 and A_1 in conjunction with $\overline{RD}$ and $\overline{WR}$. A_0 and A_1 are normally connected to the least signification bits of the address bus. If two 8255 units are used the address of ports are as follows:

For the 1st unit of 8255, ie, 8255.1:

Port / control word register	Port / control word register address
Port A	00
Port B	01
Port c	02
Control word register	03

For the 2nd unit of 8255 ie, 8255.2:

Port / control word registers	Port / control word register address
Port A	08
Port B	09
Port c	0A
Control word register	0B

If we write the instruction IN00, it means that it is for the port A of 82551. When this instruction is executed data are transferred from the port A to the accumulator. The instruction OUT 03 will transfer the content of the accumulator to the control word register of 82551. The instruction OUT 03 will transfer the content of the accumulator. OUT 0A transfer the content of the accumulator to the port C nof 8255.2 . The instruction OUT 0B transfers the content of the accumulator to the control word register of 8255.2 .

Operating Modes Of 8255:

The Intel 8255 has the following there modes of operation which are selected by software.

Mode 0 – simple input/output

Mode 1 – strobed input/output

Mode 2 – bidirectional port

The 8255 has two 8 bit ports (port A and port B) and two 4-bit port(port C_{upper} and port C_{lower}). In mode C operation C port can be operated as a simple input or output port. Each of the form ports of 8255 can be programmed to be either an input or output port. Mode 1 is strobed input/output mode of operation. The Port A and Port B both are designed to operate in this mode of operation. When Port A and Port B are programmed in mode 1, six pins of part C are used for their control. PC_o, PC_1 *and* PC_2 are used for their control of the port B which can be used either as input or output port. If the port A is operated as an input port, PC_3, PC_4 *and* PC_5 are used for this control. The remaining pairs of port c ie, PC_6 *and* PC_7 can be used as either input or output. When port A is operated as an output port, pins PC_3, PC_6 *and* PC_7 are used for its control. The pins PC_4 and PC_5 can be used either an input or output. The combination of mode 1 and mode 0 operation is also possible. For example, when port A is programmed to operate in mode 1, the port B can be operated in mode 0. Mode 2 is strobed bidirectional mode operation. In this mode port A can be programmed to operates as a bidirectional port. The mode 2 operation is only for port A. when port A is programmed to mode 2, the port B can be used in either mode 1 or mode 0 for mode 2 operation PC_3 *to* PC_7 are used for the control of port A.

<u>Control Groups:</u>

A control word is formed which contains the information regarding the function and mode of the ports.

Control word:

According to the requirement, a port can be programmed to act either as an input port or an output port. For programming the ports of 8255 a control word is formed. The bits of control word are shown in the Fig.3.12.

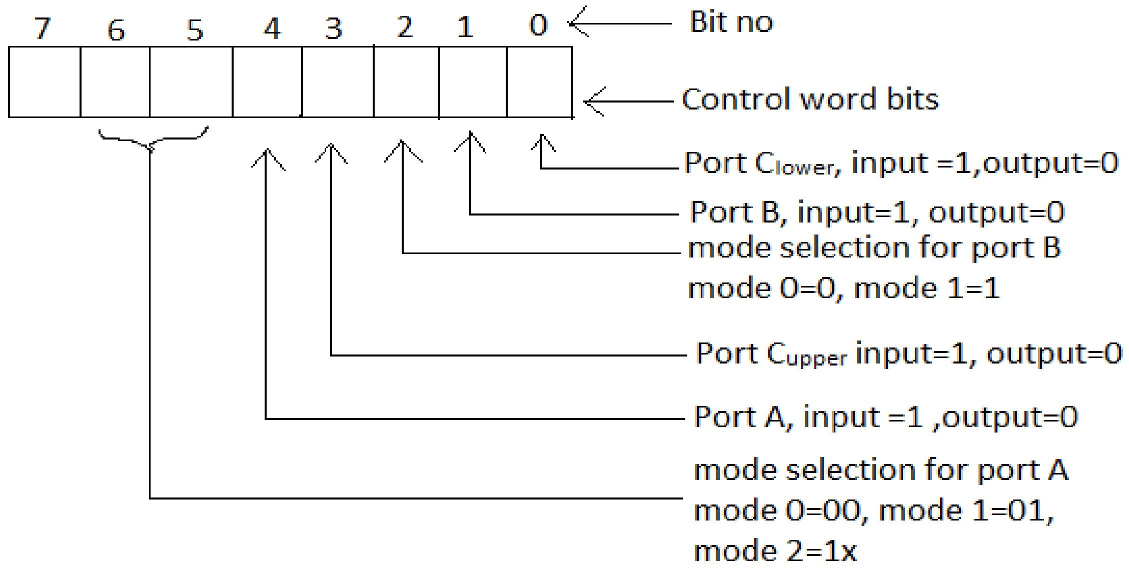

7 6 5 4 3 2 1 0
Bit no
Control word bits
Port Clower, input =1,output=0
Port B, input=1, output=0
mode selection for port B
mode 0=0, mode 1=1
Port Cupper input=1, output=0
Port A, input =1 ,output=0
mode selection for port A
mode 0=00, mode 1=01,
mode 2=1x

Fig.3.12 Control word bits for Intel 8255

Control word is written into the control word registers which is within 8255. For control word register which is with in 8255. For control word register only OUT instruction is used. It contains cannot be read. The control word bit corresponding to a particular port is set to either 1 or 0 depending upon the definition of the port, whether it is to be made an input port or output port. If a particular port is to be made an input port, the bit corresponding to that port is set to 1. For making a port an output port, the corresponding bit for the port is set to 0. The detained description of the bits of the control word is as follows:

Bit No 0:

It is for port c_{lower}

To make port C_{lower} an input port, the bit is set to 1.

To make port C_{lower} an output port, the bit is set to 0.

Bit No 1:

It is for port B

To make port B and input port, the bit is set to 1.

To make port b an output port, the bit is set to 0.

Bit No 2:

It is for the selection of the mode for the port B. if the port B has to operate in mode 0, the bit is set to 0. For mode 1 operation of the port B, it is set to 1.

Bit No 3:

It is for the port C_{upper}

To make port C_{upper} an input port, the bit set to 1.

To make port C_{upper} an output port, the bit is set to 0.

Bit No 4:

It is for the port A.

To make portA, an input port, the bit set to .

To make port A, an output port, the bit is set to 0.

Bit No 5 and 6: These bits are to define the operating mode of the port A. For the various modes of port A these bits can defined as follows:

Mode of Port A	Bit no 6	Bit no 5
Mode 0	0	0
Mode 1	0	1
Mode 2	1	0 or 1

For mode 2 bit no 5 is set either 0 or 1 it is inmaterial.

Bit No 7: It is set to 1 if ports A,B and C are defined as input/output ports. It is set to 0 if the individual pins of the port C are to be set or reset.

Examples:

The following examples will illustrate how to make control words.

Ex:1

Make control word when the ports of Intel/8255 are defined as follows

Port A as an input port.

Mode of the port A – mode 0

Port B as an output port.

Mode of the port B – mode 0.

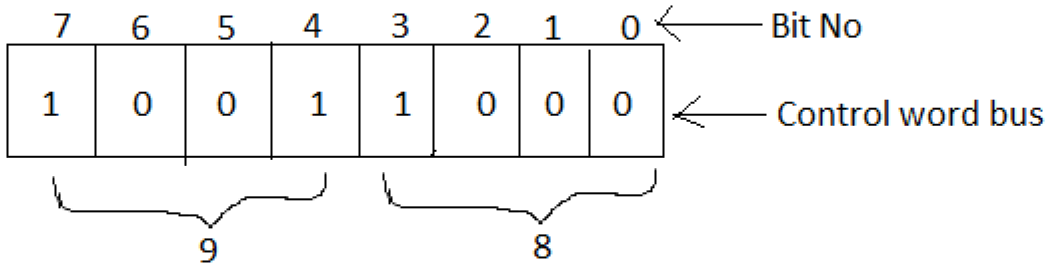
Mode of the port B – mode 0

Port C_{upper} as an input port

Port C_{lower} as an output port

Solution:

The control word bits for the above definition of the ports are as follows:



The control word =98H

Bit No.0 is set to 0, as the port C_{lower} is an output port.

Bit No.1 is set to 0, as the port B is an output port.

Bit No.2 is set to 0, as the port B has to operate in mode 0.

Bit No.3 is set to 1, as the port C_{upper} is an input port.

Bit No.4 is set to 1 , as the port a is an input port.

Bit No5 and 6 are set to 00 as the port A has to operate in mode 0.

Bit No 7 is set to 1, as the ports of A ,B and C are used as simple input/output port.

Thus the control word= 98H.

3.13 Programmable DMA controller:

The direct memory access (DMA) data transfer schemes, data are directly transferred from I/O device to RAM or from RAM to I/O devices. The DMA data transfer , the data and address buses come reads the control of the peripheral devices which wants DMA data transfer. The µp has to relinquish to control of the address and data buses for DMA operation on the request of I/O devices. For DMA data transfer the I/O device must have its own register. It most also be able to generate control signals required for DMA data transfer. Generally such facilities are not available with I/o devices. Single chip programmable DMA controllers have been developed by several manufacturers for the interfacing of I/O devices to the µp for DMA data transfer. Such controller provide all the facilities for DMA data transfer. Intel 8257 and 8237 are the most examples of DMA controllers.

Intel 8257:

It is a programmable DMA controller. Figure 3.13 shows its schematic diagram. It is a 4-channel programmable direct memory access (DMA) controller. It is a 40-pin I.C. package and requires a single +5v supply for its operation. Four I/o devices can be interfaced to the µp through this device. It is capable of performing three operation namely read, write and verify. During the read operation data are transferred form the memory to the directly transferred form the memory to the I/O device. During the write operation data are transferred from the I/o device to the memory on receiving a request from the I/O device, the 8257 generator a sequential memory address which allows the I/O device, to read or write directly to or from the memory. Each channel incorporates two 16- bit resisters namely,

1. DMA address register
2. Byte count register

These register are initialized before a channel is enable. Initially the DMA address of the first memory location to be accessed. During DMA operation it stores the next memory locations to be accessed in the next DMA cycle. 14-LSBs of the byte count registers store the number of bytes to be transferred. 2^{14}(16384) bytes of data can directly be transferred to the memory from the I/O device or from the memory to the I/O device. 2 MSBs of the byte count register indicate the operation which will be performed by the controller on that channel. Besides these register the 8257 also includes of mode set register and starter register.

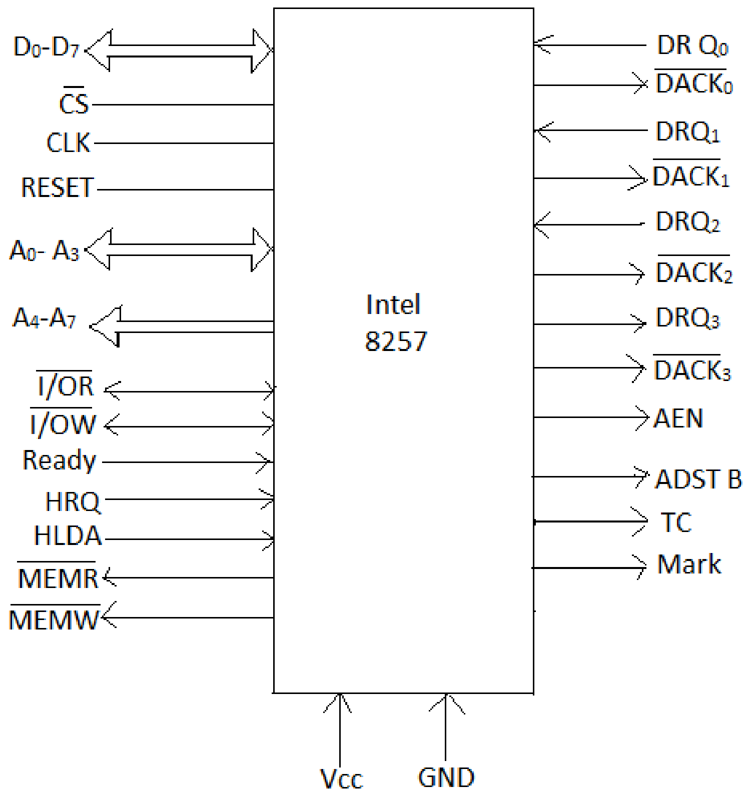


Fig.3.13 Schematic diagram of Intel 8257

The important pins of Intel 8257 are as follows:

$\boldsymbol{DRQ_0 - DRQ_3}$: These are DMA request lines. An I/O device sends its DMA request on one of these lines. A high status of the line generator a DMA request.

$\overline{\boldsymbol{DACK_0}} - \overline{\boldsymbol{DACK_3}}$: These are the DMA acknowledge lines. The Intel 8257 sends an acknowledge signal through one of these lines informing an I/O device that it has been selected for DMA data transfer. A low on the line acknowledge the I/O devices.

$\boldsymbol{A_0 - A_7}$**:** These are address lines. $A_0 - A_3$ are bidirectional lines. In the master mode these lines carry 4 LSB of 16- bit memory address generated by the 8257.ln the slave mode these lines are input lines. The input select one of the register to be read or programmed. $A_4 - A_7$ lines give tristated outputs which carry 4 through 7 of the 16- bit memory address generated by the 8257.

$\boldsymbol{D_0 - D_7}$**:** These are data lines. These are bi-directional three state lines. While programming the controller the CPU sends data for the DMA address register, the byte count register, and mode set register through these data lines. During DMA cycle, the 8257 sends the 8 MSBs are then latched in 8212 latch. Therefore the data bus is made available to handle memory data transfer during reset of the DMA cycle.

$\boldsymbol{AEN}$**:** Address enable

$\boldsymbol{ADSTB}$**:** A high on this line latches the 8 MSBs of the address which are sent on D-bus into Intel 8212 connected for this purpose.

$\boldsymbol{\overline{CS}}$**:** It is a chip select.

$\boldsymbol{\overline{I/OR}}$**:** I/O read. It is a bidirectional line. For output mode it is used to access data from the I/O device during the DMA write cycle.

$\boldsymbol{\overline{I/OW}}$**:**I/O write. It is bidirectional line. In output mode it allows the transfer of data to the I/O device during the DMA read cycle. Data is transferred from the memory.

$\boldsymbol{\overline{MEMR}}$**:** Memory read.

$\boldsymbol{\overline{MEMW}}$**:** Memory write.

$\boldsymbol{TC}$**:** Byte count (terminal count)

$\boldsymbol{MARTC}$**:** Modulo 128 mark

$\boldsymbol{CLK}$**:** Clock.

$\boldsymbol{HRQ}$**:** Hold request

$\boldsymbol{HLDA}$**:** Hold acknowledge.

An I/O device sends its request for DMA transfer through one of the four DRQ lines. On receiving the DMA request for DMA data transfer from an I/O device, the Intel 8257 sends the hold request to the CPU through the HRQ lines. The 8257 receives the hold acknowledge signal from the CPU through HLDA line. After receiving the HOLD acknowledge from CPU, it sends DMA acknowledge to the I/O device through $\overline{DACK}$ line. The memory address is sent out on address and data lines. The 8257 sends 8 MSBs of the memory address over D-bus. These 8MSBs of the memory address are latched into 8212 using ADSTB signal. ADSTB is similar to ALE of Intel 8085. For DMA send cycle, in which data are transferred from memory to I/O devices. Two control signal $\overline{MEMR}$ and $\overline{I/OW}$ are issued by 8257. The $\overline{MEMR}$ enables the addressed memory for reading data from it. The $\overline{I/OW}$ enables the I/O device to accept data. similarly, for DMA write cycle in which data are transferred from the I/O device to the memory, two control signals $\overline{MEMW}$ and $\overline{I/OR}$ are issued by the controller. The $\overline{MEMW}$ enables the addressed memory for writing data to it.

The $\overline{I/OR}$ enables the I/O device to output data. The byte count is decremented by one after the transfer of one byte of data, when byte count becomes zero, TC goes high indicating that the data using DMA is complete in a fixed priority mode or rotating mode of operation. READY line is used by slow memory or I/O devices.

3.14 Programmable Interrupt Controller(PIC):

The programmable interrupt controllers is used when several I/O devices transfer data using interrupt and they are to be connected to the same interrupt level of the µp. when the number of the I/O devices is less than the number of interrupt levels of the µp, such controllers are not required. The Intel 8259 is a single chip programmable interrupt controller. It is compatible with 8086, 8088 and 8085 microprocessor. It is a 28 pin DIP IC package and uses N-Mos technology. It requires a single +5V supply for its operation. Figure 3.14 shows the schematic diagram of Intel 8259.

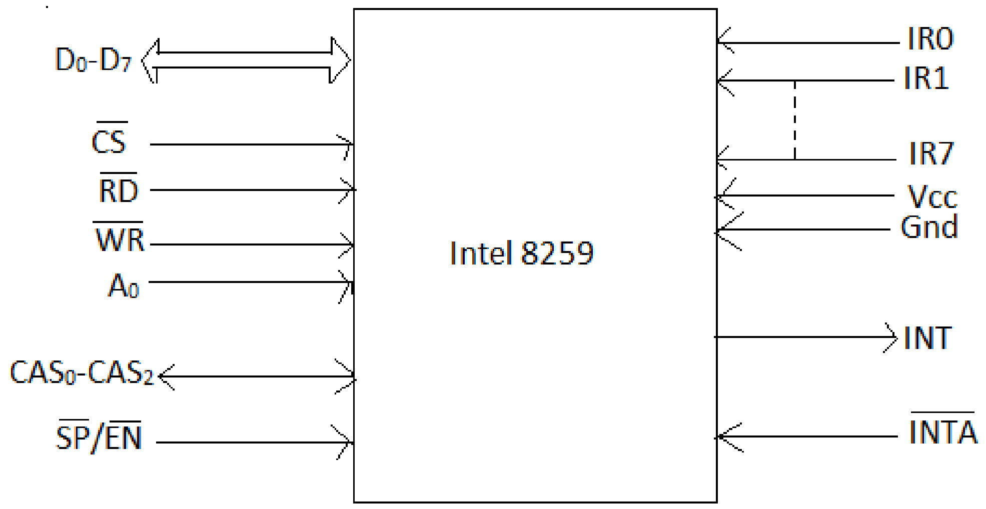


Fig.3.14 Schematic diagram of Intel 8259

The detail of its pins are as follows:

$\overline{\boldsymbol{CS}}$**:** Chip select

$\overline{\boldsymbol{WR}}$**:** Write A low on this pin enables Intel 8259 to accept command word CPU.

$\overline{\boldsymbol{RD}}$**:** Read. A low on this pin enables Intel 8259 to send the various status signals on the data bus for CPU.

$\boldsymbol{P_0 - P_7}$**:** Bidirectional data bus control, status and interrupt vector information are transformed via this bus.

$\boldsymbol{CAS_0 - CAS_2}$**:** cascade lines.

$\boldsymbol{\overline{SP}/\overline{EN}}$**:** Slave programmable/ enable buffer

$\boldsymbol{INT}$: Interrupt . It is used to interrupt CPU.

$\boldsymbol{\overline{INTA}}$**:** Interrupt acknowledge.

$\boldsymbol{IR_0 - IR_7}$**:** Interrupt request. I/O devices send interrupt request these lines.

$\boldsymbol{A_0}$**:** Address line. It acts in configuration with $\overline{RD}$,$\overline{WR}$ and $\overline{CS}$. The Intel 8259 uses it to interrupt command words the CPU writes and status the CPU wants to send.

The following Fig. 3.15 shows the block diagram of connections of PIC and I/O devices to the micro computers.

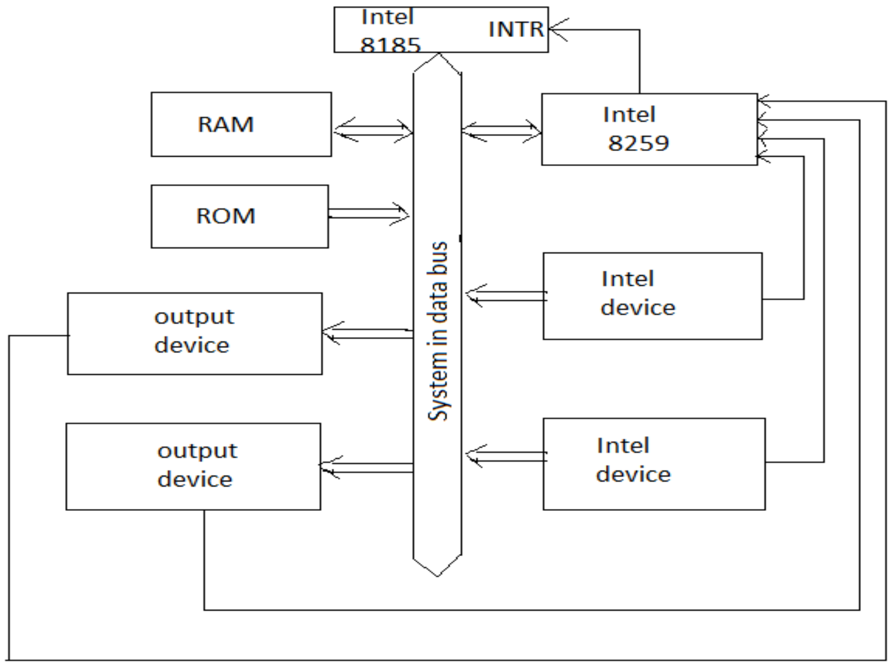


Fig.3.15 Interfacing of 8259 and I/O devices

The priority can be assigned to the I/O devices connected to PIC. 8 I/O devices can be connected to 8259 through $IR_0 - IR_7$ lines. The interrupt controller functions as overall manages is an interrupt driven system. It accepts request from an I/O device and determines which of the incoming request is of the highest priority. After checking the priority of the interrupt request, the 8259 sends an interrupt signal to the µp through the INT line. The µp sends the acknowledge signal through $\overline{INTA}$ line can receiving $\overline{INTA}$ signal all the interrupt of lower priority are inhibited and the 8259 sends a CALL instruction to the µp. the CALL instruction is unique so that the µp can take up the ISS for the I/O device which has requested for data transfer. 8259 chips can be cascaded to receive upto 64 vectored priority interrupts without additional circuitory.

3.16 Programmable Counter/ Timer- Intel 8253:

When the processor has to perform time based activities, there are two methods to the processor can execute a delay subroutine. In this method, the delay subroutine will load a count value in one of the register of the processor and start's decrementing the count value. After every decrement operation, the zero flag is checked to verify whether, the count has reached zero or not. If the count has reached zero the delay subroutine is terminated. Now the desired time will be elapsed and the processor and perform the desired time based task. In this method, the time is estimated interms of processor clock periods to execute the delay subroutine.

In second method an external times can maintain the timings and interrupt the processor at periodic intervals. In the first method the processor time is wasted by simply decremented a register. But in the second method, the processor time can be efficiently utilized, because the processor can perform other tasks in between times interrupts.

Popular programmable internal timer chips are Intel 8253 and 8254. The 8253 operator in the frequency range of d.c to 10 MHz. The 8253 uses NMOS technology where as 8254 HMOS technology. Both are pin to pin compatible and operation in the following six modes.

Mode 0: Interrupt on terminal count

Mode 1: Programmable one shot

Mode 2: Rate generator

Mode 3: square wave mode

Mode 4: software trigged mode

Mode 5: Hardware trigged mode.

The 8254 is compatible to 8086, 8086, 8085 and most µps. The 8253 is compatible to 8085 µp. the 8254 is a super set 8253.

<u>Intel 8253:</u>

The 8253 is 24 pin IC and operation at 5V d.c. It contains three independent 16-bit counters, which can be programmed to work is any one of the possible six modes. Each counter has a clock input, gate output and counter output. To operate a counter, a count value has to be loaded in count register, gate should be tied high and a clock signal should be applied through clock input. The pin configuration of 8253 is shown in Fig.3.16. The functional block diagram of 8253 is shown in Fig.3.17.

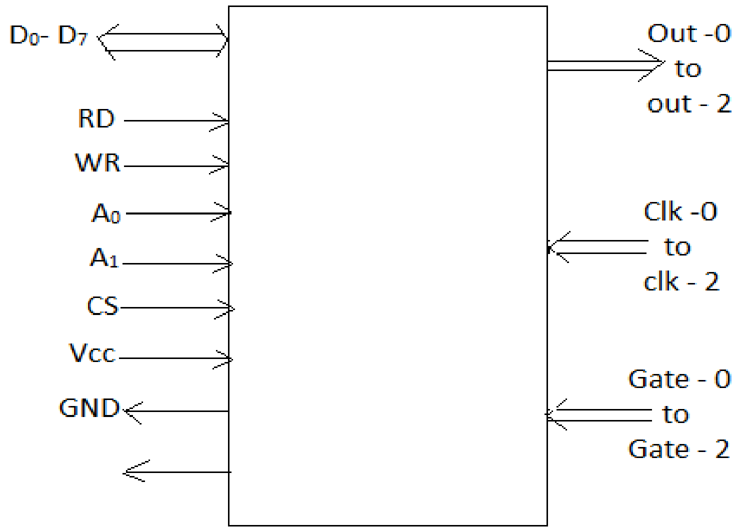
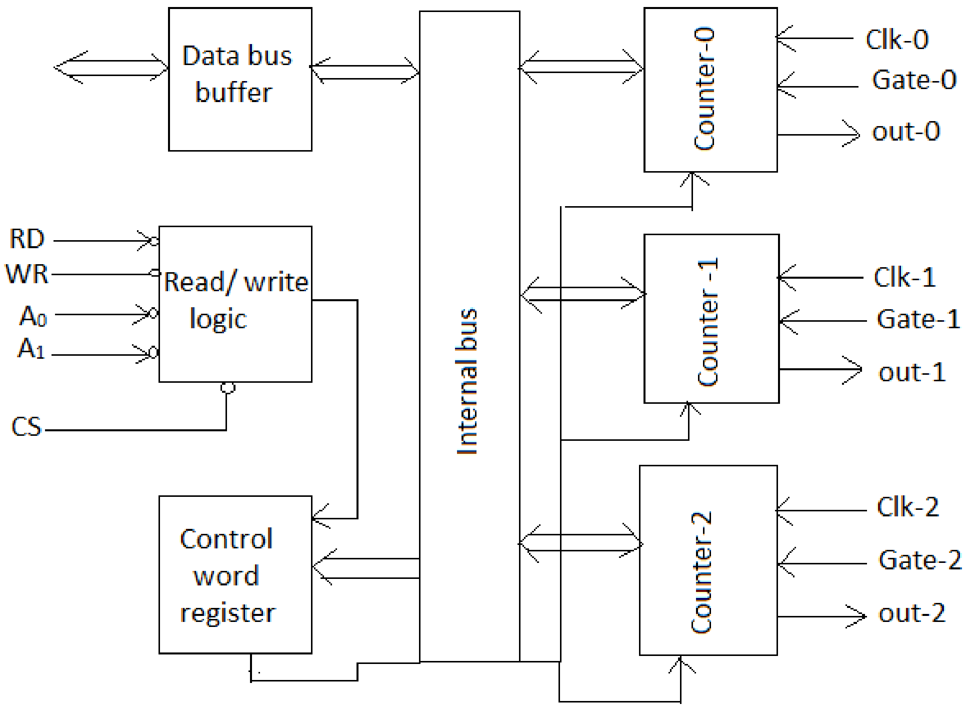


Fig.3.16 The pin configuration of 8253



3.17 The functional block diagram of 8253

Pin	Description
D_0- D_7	Bi directional data bus.
$\overline{CS}$	Chip select
$\overline{RD}$	Read control
$\overline{WR}$	Write control
A_0, A_1	Internal address
Clk -0 to clk – 2	Clock input to counters
Gate -0 to gate – 2	Gate controls input counters
Out-0 to out -2	Output of counters

The 8253 has eight data lines which can be used for communication with processor. The control words and count values are written in 8253 registers through data bus buffer. The $\overline{CS}$ is used to select the chip. The address lines A_0 *and* A_1 are used to select any one of the four internal devices as shown in table 3.4:

Table 3.4: Internal address of 8253

Internal address		Device selected
A_0	A_1	
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control register

The control register $\overline{RD}$ and $\overline{WR}$ are used by the processor to perform read/ write operation.

Control word register:

When the pins A_0, A_1 are 11, the control word register is selected. The control word format is shown below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RL1	RLO	M_2	M_1	M_0	BCD

The bits D_7 *and* D_6 of the control word are to select one of the 3 counters. D_5 *and* D_4 are for loading/reading the count. D_3, D_2 *and* D_1 are for the selection of operating mode of the selected counter. There are six modes of operation for each counter of 8253. The selected counters can be programmed to operate in any desired mode. The six modes of operation are: mode 0, mode 1, mode 2, mode 3, mode 4 and mode 5. The bit D_0 for the selection a binary of BCD counting. When D_0 is set to 0 the selected counter operator as a binary counter. When it is set to 1 the counter operator as a BCD counter.

The function of counter of 8253 are programmed by the system software. The control word decides the selection of counters, its mode of operation, loading sequence of the count and selection of binary or BCD counting. As soon as the control word is written into the control word register the desired counter is selected, mode of operation is set and the loading sequence of the count and the selection of binary or BCD counting are defined.

SC(Select counter): To select the counters SC_0 and SC/ are set as follows:

SC_1	SC_0	
0	0	Select counter 0
0	1	Select counter 1
1	0	Select counter 2
1	1	Illegal

RL(Read/ load): To load read counts RL_0 *and* RL_1 are set as follows:

RL_1	RL_0	
0	0	Counter latching operation
0	1	Read/ Load Least signification byte only
1	0	Read/ Load most signification byte only
1	1	Read/ Load signification byte first, then most signification byte.

Mode: Mode selecting bits M_0, M_1 *and* M_2 are set as follows:

M_2	M_1	M_0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	0	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD:

0	Binary counter 16 bits.
1	Binary coded decimal (BCD) counter(4 decodes)

<u>Reading while counting:</u>

There is a commend for latching the content of the counts to lead its count while count is still going on. The bit pattern for the control word for latching operation as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC_1	SC_0	0	0	X	X	X	x

SC_1 and SC_0 – specified counter to be latched.

D_5 and D_4 – 00 makes counter latching operation

X - indicates don't care.

A counter can be used for various applications such as BCD/Binary counter, programmable rate generator, square wave generator, hardware/ software triggered strobe, programmable one shot, to generate time delay etc.

1) *<u>Mode 0: Interrupt on terminal count</u>*<u>:(</u> to generate accurate time delay)

In this mode, the timer is loaded with a value, the counter decrements for each clock pulse. When the counter reaches zero, an interrupt is generated.

In this mode; initially the OUT is low, once a count is loaded in the register, the counter is decremented every cycle, and when the count reaches zero, the OUT goes high. This is can be used as an interrupt. The OUT remains high until a new count or command word is loaded. Figure 3.18 shown the timing diagram for mode 0 operation.

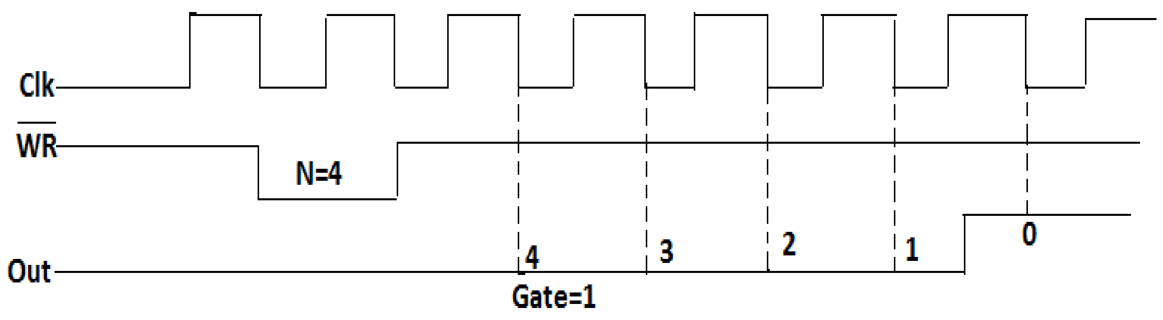
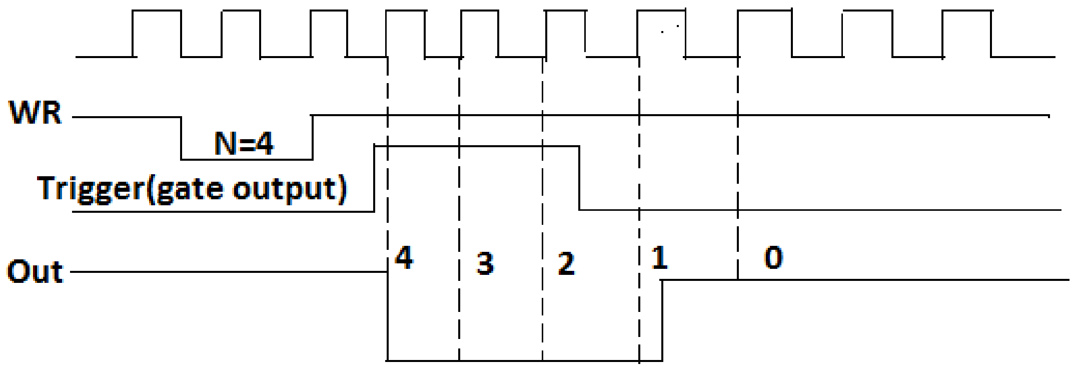


Fig. 3.18

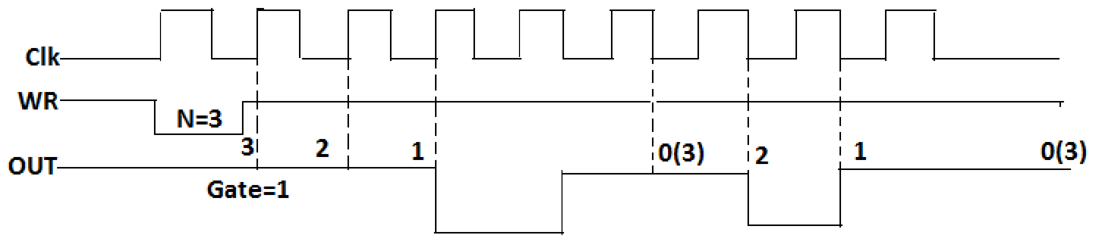
Mode 1: programmable one shot: A pulse length and initial counter is set before the first cycle, when the counter reaches zero, the output goes high and stays high until reset. This process is repeated each time it is trigged by the trigger pulse. In this mode, the out is initially high. When the Gate is triggered, the out goes low and at the end of the count, the out goes high again, thus generating a one-shot pulse.

The width of the output pulse can be varied by varying N. As the width of the output pulse is programmable this mode of operation is known as programmable one shot.



Mode 2: Rate generator:

In mode 2 the counter act as a simple divide by N counts. This mode is used to generate a pulse equal to the clock period at a given internal. When a count is loaded, the OUT stays high until the count reaches 1, and then the OUT goes low for one clock period A_6 in this the output becomes high again and the count N is automatically reloaded into the counters.

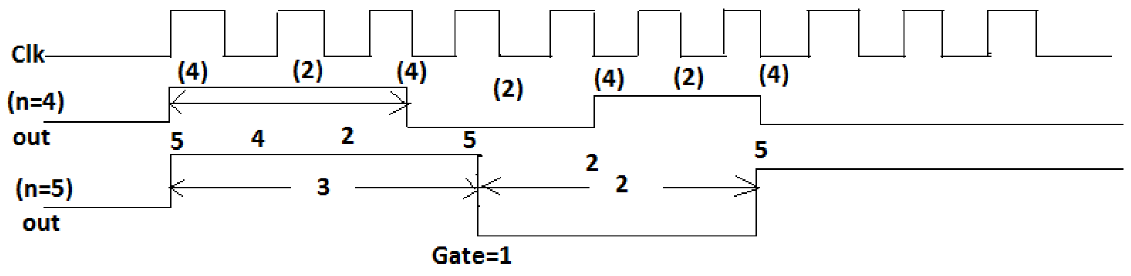


Mode 3: Square wave generator:

Here, a square wave is generated by the timer in this mode. The counter output remains high during the first half number of clock pulses and low during the other half number of clock pulses. When the counter reaches zero, the process is repeated.

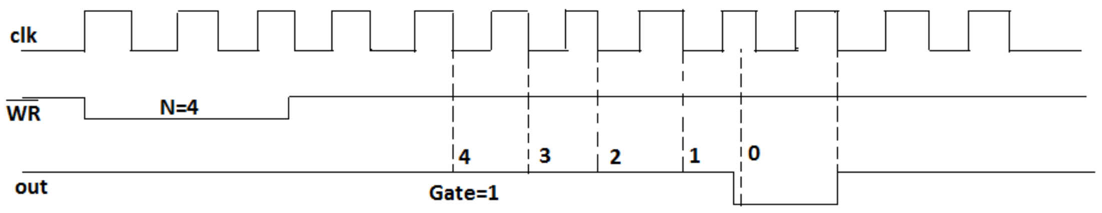
In this mode, when a count is loaded, the OUT is high. The count is decremented by two at every cycle, and when it reaches zero, the OUT goes low, and the count is reloaded again. This is repeated continuously, thus a continuous square wave with period equal to the period of the count is generated. In other words, the frequency of the square wave is equal to the frequency of

the clock is divided by the count. If the count (N) is odd, the pulse stays high for (N+1
)/2 clock cycles and stays low for (N-2)/2 clock cycles. If the count is even, the pulse stays high for N/2 and stays low for N/2 clock cycles.



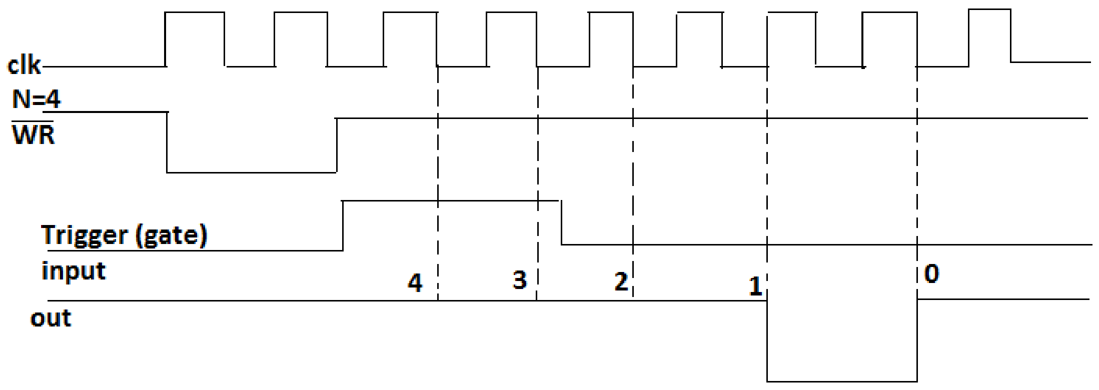
Mode 4: Software Triggered strobe:

The timer is loaded with a value and the value is counted down to zero; one count per clock pulse. When the counter reaches zero, a pulse is generated. A software commend trigger or status the next cycle.



Mode 5: Hardware Triggered strobe:

The timer is loaded with a value and the value is counted down to zero. One count per clock pulse. When the counter reaches zero, a pulse is generated. A hardware command triggers or starts the next cycle.



UNIT-IV

Programming of 8086

4.1 8086 CPU Architecture

The internal functions of the 8086 processor are partitioned logically into two processing units as shown in Fig.4.1.

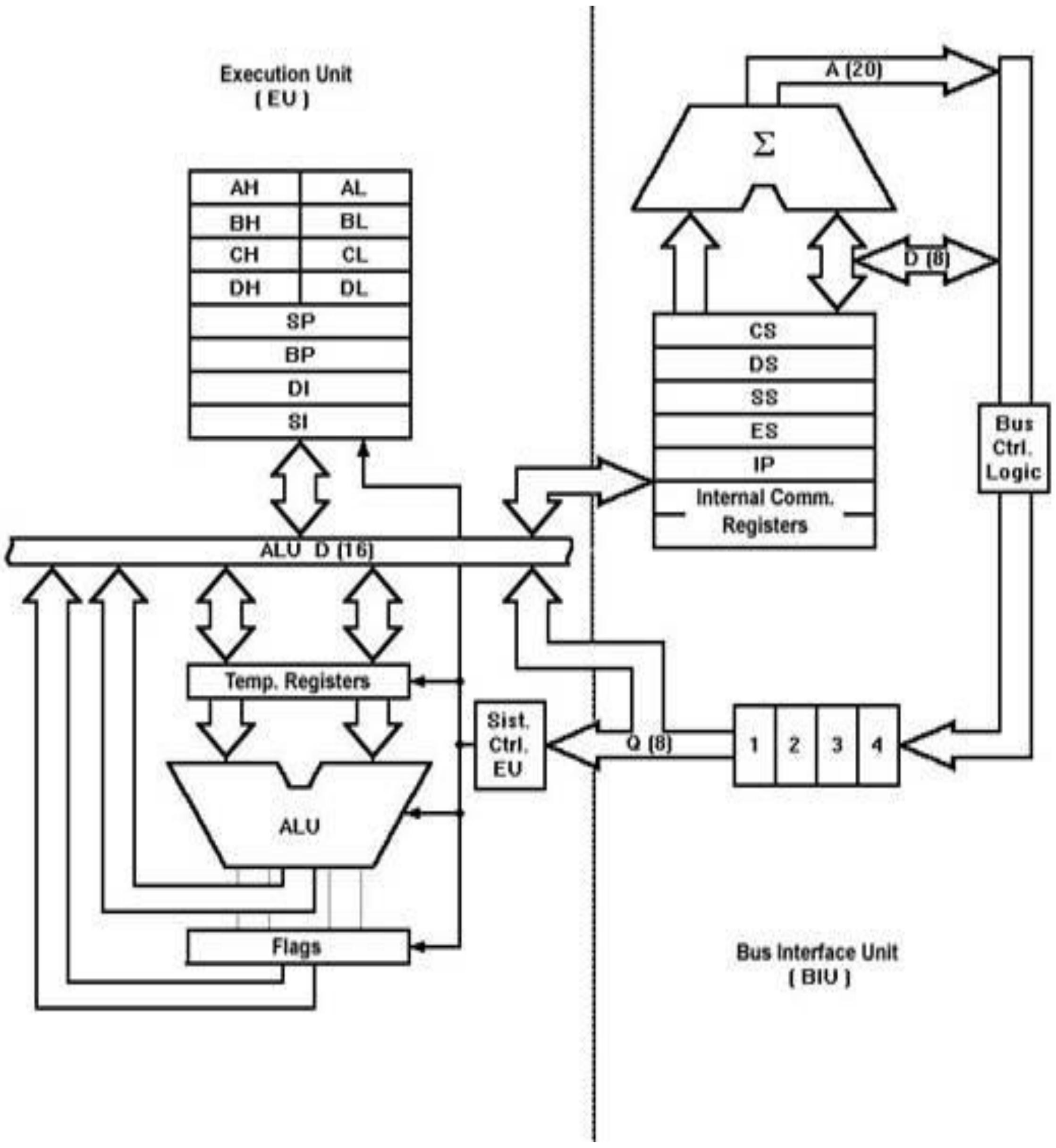


Fig.4.1 Architecture of 8086

8086 microprocessor has two units; Execution Unit (EU) and Bus Interface Unit (BIU). They are dependent and get worked by each other. Below is a short description of these two units.

Execution Unit (EU):

The EU contains (i) ALU (ii) General purpose registers (iii) Index registers (iv) pointers .

Execution unit receives program instruction codes and data from the BIU, executes them and stores the results in the general registers. It can also store the data in a memory location or send them to an I/O device by passing the data back to the BIU. This unit, EU, has no connection with the system Buses. It receives and outputs all its data through BIU.

ALU (Arithmetic and Logic Unit) : The EU unit contains a circuit board called the Arithmetic and Logic Unit. This unit can perform various arithmetic and logical operation, if required, based on the instruction to be executed. It can perform arithmetical operations, such as add, subtract, increment, decrement, convert byte/word and compare etc and logical operations, such as AND, OR, exclusive OR, shift/rotate and test etc.

Registers : A register is like a memory location where the exception is that these are denoted by name rather than numbers. It has 4 data registers, AX, BX, CX, DX and 2 pointer registers SP, BP and 2 index registers SI, DI and 1 temporary register and 1 status register FLAGS .

AX, BX, CX and DX registers has 2 8-bit registers to access the high and low byte data registers. The high byte of AX is called AH and the low byte is AL. Similarly, the high and low bytes of BX, CX, DX are BH and BL, CH and Cl, DH and DL respectively. All the data, pointer, index and status registers are of 16 bits. Else these, the temporary register holds the operands for the ALU and the individual bits of the FLAGS register reflect the result of a computation.

Bus Interface Unit:

The BIU contains (i) Segment registers (ii) Instruction registers (iii) Instruction queue and (v) Flag register

As the EU has no connection with the system Busses, this job is done by BIU. BIU and EU are connected with an internal bus. BIU connects EU with the memory or I/O circuits. It is responsible for transmitting data, addresses and control signal on the busses.

Registers : BIU has 4 segment busses, CS, DS, SS, ES. These all 4 segment registers holds the addresses of instructions and data in memory. These values are used by the processor to access memory locations. It also contain 1 pointer register IP. IP contains the address of the next instruction to executed by the EU.

Instruction Queue : BIU also contain an instruction queue. When the EU executes instructions, the BIU gets up to 6 bytes of the next instruction and stores them in the instruction queue and this process is called instruction prefetch. This is a process to speed up the processor. Also when the EU needs to be connected with memory or peripherals, BIU suspends instruction prefetch and performs the needed operations

Purpose of using Instruction Queue:

BIU contains an instruction queue. When the EU executes instructions, the BIU gets up to 6 bytes of the next instruction and stores them in the instruction queue and this process is called instruction prefetch. This is a process to speed up the processor. A subtle advantage of instruction queue is that, as next several instructions are usually in the queue, the BIU can access memory at a somewhat "leisurely" pace. This means that slow-memory parts can be used without affecting overall system performance.

5.2 Instruction Set of Intel 8086 Microprocessor

Instruction set of 8086 microprocessor can be divided into data copy/transfer instructions, arithmetic and logical instructions, branch/loop instructions, machine control instructions, flag manipulation instructions, string manipulation instructions. Instruction set refers to the instructions which can be used to program a microprocessor etc,. Instruction set can be divided into data copy/transfer instructions, arithmetic and logical instructions, branch/loop instructions, machine control instructions, flag manipulation instructions, string manipulation instructions. The instruction set of 8086 microprocessor is:

(i) Data Copy/Transfer Instructions

These are the type of instructions used to copy, move etc., data from source to destination. Some of the data copy/transfer instructions are:

MOV	:	Move data from register to register, memory to register, register to memory, memory to accumulator, accumulator to memory etc,.
PUSH	:	Push data into register, memory etc,.
POP	:	Pop data from register, memory etc,.
XCHG	:	Exchange data between register, memory etc,.
IN	:	Input from fixed port or variable port
OUT	:	Output to fixed port or variable port
LDS	:	Load pointer to DS
LES	:	Load pointer to ES
LAHF	:	Load AH with flags
SAHF	:	Store AH into flags
PUSHF	:	Push flags
POPF	:	Pop flags

(ii) Arithmetic and Logical Instructions

These are the type of instructions used to perform arithmetic operations like addition, subtraction etc., and logical operations like and, or etc,. Some of the arithmetic and logical instructions are :

(a) Arithmetic Instructions

ADD	: Addition
ADC	: Addition with Carry
INC	: Increment by 1
AAA	: ASCII Adjust for Addition
DAA	: Decimal Adjust for Addition
SUB	: Subtraction
SBB	: Subtraction with Borrow
DEC	: Decrement by 1
AAS	: ASCII Adjust for Subtraction
DAS	: Decimal Adjust for Subtraction
MUL	: Unsigned Multiplication
IMUL	: Signed Multiplication
AAM	: ASCII Adjust for Multiplication
DIV	: Unsigned Division
IDIV	: Signed Division
AAD	: ASCII Adjust for Division
NEG	: Change Sign
CMP	: Compare
CBW	: Convert Byte to Word
CWD	: Convert Word to Double Word

(b) Logical Instructions

AND	: Logical AND
OR	: Logical OR
NOT	: Logical NOT
XOR	: Logical XOR
SHL	: Shift Logical Left
SHR	: Shift Logical Right
ROL	: Rotate Left
ROR	: Rotate Right
RCL	: Rotate Left through Carry Flag
RCR	: Rotate Right through Carry Flag

(iii) Branch/Loop Instructions

These are the type of instructions used to control the transfer to a specified address. Some of the branch/loop instructions are:

(a) Unconditional Branch/Loop Instructions

CALL	: Call a subroutine Unconditionally
RET	: Return from a procedure

INTN : Interrupt of Type N
INTO : Interrupt on Over flow
LOOP : Loop instructions Unconditionally

(b) Conditional Branch/Loop Instructions

JZ : Jump if zero
JE : Jump if equal
JNZ : Jump if not zero
JNE : Jump if not equal
JL : Jump if lesser
JLE : Jump if lesser or equal
JG : Jump if greater
JGE : Jump if greater or equal
JO : Jump on Over flow
JNO : Jump on not Over flow
JS : Jump on Sign
JNS : Jump on not Sign
LOOPZ : Loop if zero
LOOPE : Loop if equal
LOOPNZ : Loop if not zero
LOOPNE : Loop if not equal

(iv) Machine Control Instructions

These are type of instructions used to control machine status. Some of the machine control instructions are:

WAIT : Wait for the test input to go low

HLT : Halt the processor

NOP : No operation

ESC : Escape to external device

LOCK : Lock instruction prefix

(v) Flag manipulation Instructions

These are the type of instructions used to manipulate different flags present in the flag register of 8086 microprocessor. Some of the flag manipulation instructions are:

CLC : Clear Carry Flag
STC : Set Carry Flag
CLD : Clear Direction Flag
STD : Set Direction Flag

CLI : Clear Interrupt Flag
STI : Set Interrupt Flag

(vi) String Manipulation Instructions

These are the type of instructions used to manipulate strings. Some of the string manipulation operations are:

REP	: Repeat Instruction Prefix
REPE	: Repeat if equal
REPZ	: Repeat if zero
REPNE	: Repeat if not equal
REPNZ	: Repeat if not zero
MOVS	: Move String Byte/Word
CMPS	: Compare String Byte/Word
SCAS	: Scan String Byte/Word
LODS	: Load String Byte/Word
STOS	: Store String Byte/Word

5.3 Addressing Modes of 8086

8086 memory addressing modes provide flexible access to memory, allowing you to easily access variables, arrays, records, pointers, and other complex data types. 12 addressing modes classified in 5 groups

(1) Addressing modes for register and immediate data

(i). Register addressing: - the instruction will specify the name of the register which holds the data to be operated by the instruction

Ex: MOV CL, DH : content of 8-bit DH register is moved to another 8-bit register CL

(ii). Immediate addressing: - an 8-bit or 16-bit data is specified as a part of the instruction

Ex: MOV DL, 08H : The 8-bit data (08H) given in the instruction is moved to DL register

(2) Addressing modes for memory data

(iii). Direct addressing: - an unsigned 16-bit displacement or signed 8-bit displacement will be specified in the instruction

Ex: MOV DX, [08H] EA=0008H (sign extended 8-bit displacement)
BA= (DS)*16; MA=BA+EA
(DX)<= (MA) OR DL<= (MA)
DH<= (MA+1)

(iv). Register indirect addressing: - the name of the register which holds the effective address will be specified in the instruction

-Ex: MOV CX, [BX]

(v*). Based addressing*: - BX or BP register is used to hold a base value for effective address and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction

-Ex: MOV AX, [BX+08H]

(vi). Indexed addressing: -SI or DI reg. is used to hold an index value for memory data and a signed 8-bit displacement or unsigned 16-bit displacement will be specified in the instruction.

-Ex: MOV CX,[SI+0A2H]

(vii). Based index addressing: - the EA is given by the sum of base value and 8-bit or 16-bit displacement specified in the instruction

-Ex: MOV DX,[BX+SI+0AH]

(viii). String addressing: - for string instructions - EA of source data is stored in SI reg. and the EA of destination data is stored in DI reg.

(3) Addressing modes for I/O ports

(ix). Direct I/O port addressing: - used to access data from standard I/O mapped devices or ports.

-Ex: IN AL, [09H]: the content of port with address 09H is moved to AL

(x). Indirect I/O port addressing: - is used to access data from standard I/O mapped devices or ports and the instruction will specify the name of the register which holds the port address.

-Ex: OUT [DX], AX : the content of AX is moved to port whose address is specified by DX

(4) Relative addressing modes

(xi). Relative addressing: - EA of the pgm instruction is specified relative to IP by an 8-bit displacement

-Ex: JZ 0AH

(5) Implied addressing modes

(xii). Implied addressing: - in this the instruction itself will specify the data to be operated by the instruction

-Ex: CLC: clear carry

8086 Assembly Language Programs :

1. Addition of two 16-bit data:

Label	Mnemonics	comments
	MOV AX, DATA 1	Load the first data in AX register
	MOV CL, 00H	Clear the CL register for carry
	ADD AX, DATA 2	Add 2nd data to AX, sum will be in AX
	MOV 2000H, AX	Store sum in memory location 1
	JNC STEP	Check the status of carry flag
	INC CL	If carry is set: increment CL by one
STEP	MOV 2002H CL	Store carry in memory location 2
	HLT	Halt

2. Subtraction of Two 16-bit data

Label	Mnemonics	comments
	MOV SI, 2000H	Load the address of data in SI register
	MOV AX, [SI]	Get the minuend in AX register
	MOV BX, [SI+2]	Get the subtrahend in BX register
	MOV CL, 00H	Clear the CL register to account for sign
	SUB AX, BX	Get the difference in AX register
	JNC STEP	Check the status of carry flag
	INC CL	If carry is set: increment CL by one
	NOT AX	Then take 2's compliment of difference
	ADD AX, 0001H	
STEP	MOV [SI+4], AX	Store the difference in memory location 1
	MOV [SI+6], CL	Store sign bit in memory location 2

	HLT	Halt

3. Multiplication of Two 16-bit data

Label	Mnemonics	comments
	MOV AX, [2000]	Move the first data to AX register from memory
	MUL [2002]	Multiply the data in AX with the data in memory location 2002 H
	MOV [2100], DX	Save the MSW (higher order) of the result in DX Register
	MOV [2102], AX	Save the LSW (ligher order) of the result in AX register
	HLT	Halt

4. Division of Two 32-bit data by 16-bit data

Label	Mnemonics	comments
	MOV DX, [2000]	Move the high order word dividend to DX register
	MOV AX, [2002	Move the lower order word dividend to AX register
	DIV [2004]	Divide the data in DX: AX by the divisor
	MOV [2100], AX	The quotient is stored in AX
	MOV [2102], DX	The remainder is stored in DX
	HLT	Halt

5. Find the sum of the elements in an array

Label	Mnemonics	comments
	MOV SI, 2000H	Set SI register as pointer for array
	MOV DI, 3000H	Set DI register as pointer for result
	MOV CL, [SI]	Set CL as count for number of bytes in array
	INC SI	Set Si to point to I- st byte of array
	MOV AX, 0000H	Set initial sum as zero
STEP 1:	DD AL, [SI]	Add a byte of array to sum

```
                 JNC    STEP2          Check for  carry flag
                 INC    AH             If carry flag is set  then increment AH
STEP 2:          INC  SI               Increment  array pointer
                 LOOP STEP 1           Repeat addition until count is zero
                 MOV   [DI], AX        Store the sum in memory
                 HLT                   Halt
```

6**. Find the largest data of signed numbers**

```
Label            Mnemonics                comments

                 MOV  AX,  @data       Initialize DS register
                 MOV   DS,  AX
                 MOV   SI 2000H        Initialize SI register
                 MOV  BX, 0000H        Initialize maximum number
                 MOV  CX,  05H         5 numbers to be processed
STEP 2:          MOV  AX, [SI]         Load number from sequence
                 CMP   BX, AX          Compare with current  maximum number
                 JCE    STEP1
                 MOV  BX, AX           Save new maximum number
                 MOV  DX, SI           Save location of maximum number
STEP 2:          INC   SI              Update pointer
                 LOOP STEP 2:          Repeat until CX > 0
                 MOV  AX, 5C00H
                 HLT                   Halt
```

7. Sorting of Data in ascending order

```
Label            Mnemonics                comments

                 MOV  AX,  @data       Initialize DS register
                 MOV   DS,  AX
                 MOV   CX,Bytes        Number of bytes used by elements
```

STEP 1	MOV DX, CX	Copy in DX register
	DEC DX	Number of comparisons
	ADD SI, DX	Point to the last element
	MOV AX, Array [SI]	Move the number to AX register
STEP 2:	CMP Array [SI-2], AX	Compare it with previous number
	JBE STEP 3	If previous number < AX, then go to step 3
	MOV DL, Array {SI-2}	Exchange the elements
	MOV Array [SI], DL	Point to previous element
	DEC SI	
	DEC DX	Decrement counter
	JNZ STEP 2	If DX ≠ 0 then repeat
STEP 3:	MOV Array [SI], AX	Exchange
	INC CX	CX = CX +1
	CMP CX, Bytes	Compare CX with number of bytes
	JBE STEP 1	If CX < number of bytes, goto step 1
	MOV AX, 5C00H	
	HLT	Halt

8. Finding Factorial of 8-bit dat

Label	**Mnemonics**	**comments**
	MOV AX, @data	Initialize DS register
	MOV DS, AX	
	MOV AL, Num	Number to AL
STEP 1	MOV CL, AL	Copy in DX register
	DEC DX	Keep a copy in CL
	MOV AH, 00	Clear upper 8 bits of accumulator
	CMP AL, 1	If number = 1 then factorial = 1
	JC LOOP 1	
	MOV BL, 2	b = 2
	MOV AL, 1	a = 1
LOOP 2:	MUL BL	a = a x b

	INC BL	b = b +1
	CMP BL, CL	Is BL < = CL
	JNA LOOP2	If yes, then do another pass
LOOP 1:	MOV Fact , AX	Fact = AX
	MOV AX, 5C00H	
	HLT	Halt

9. Converting BCD data to Binary data

Label	**Mnemonics**	**comments**
	MOV BX, 2000H	Load the address of the datain BX register
	MOV AL, [BX]	Get the BCD data in AL register
	MOV AL, DL	Copy the data in DL register
	AND DL, OFH	Mask upper nibble (10s digit)
	AND AL, FOH	Mask lower nibble (1s digit)
	MOV CL, 4	Move a count value 04 to CL
	ROR AL, CL	Rotate the upper nibble to lower nibble position
	MOV DH, 0AH	Set mutltiplier as OAH
	MUL DX	Multiply 10s digit by OAH , the product will be in AL
	ADD AL, DL	Get sum of 1s digit and product in AL
	MOV [BX+1], AL	Save binary data in memory
	HLT	Halt

10. Generation of Fibonacci series

Label	**Mnemonics**	**comments**
	MOV SI, 2000H	Initialize SI register
	MOV CX, OAX	Number of elements to be generated in CX reg.
	XOR AL, AL	Clear the accumulator
	MOV [SI], AL	Move the first number 0 to AL
	ADD AL, 01H	Add next number

```
                INC   SI                Increment pointer
                MOV  [SI], AL           Store  the next Fibonacci number in memory
                                         pointed by SI
AGAIN:         ADD   AL, [SI]           Add the contents of accumulator to [SI]
                INC   SI                Increment pointer
                MOV  [SI], AL           To store next Fibonacci number
                DEC  SI                 Decrement pointer to get  the previous number
                                         into AL
                MOV  AL,  [SI]
                INC  SI                 Increment pointer
                LOOP    AGAIN           If CX = 0 then repeat the loop
                MOV   AX, 5C00H
                HLT                     Halt
```

Unit –V

Microprocessor System Design and Applications

5.1 Delays:

Delay routines are subroutines used for maintaining the timings of various operations in µp. In control applications, certain equipment need to be ON/OFF after a specified time delay. In some applications, a certain operation has to be repeated after a specified time interval. In such cases simple time delay routines can be used to maintain the timings of the operations.   A delay routine is generally written as a subroutine (It need not be a subroutine always. It can be even a part of main program). In delay routine a count (number) is loaded in a register of µp. Then it is decremented by one and the zero flags is checked to verify whether the content of register is zero or not. When it is zero the time delay is over and the controls transferred to main program to carry out the desired operation.

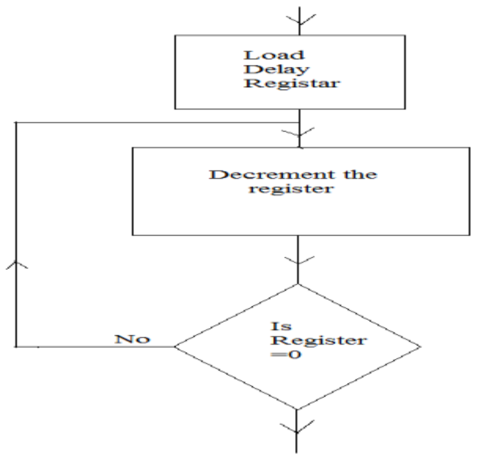
The delay time is given by the total time taken to execute the delay routine. It can computed by multiplying the total number of T required to execute subroutine and the time for T-state of the processor. The total number of time can be computed from the knowledge of T states each instruction. The time for one T state processor is given by the inverse of the internal clock frequency of the processor. For example, of the 8085 µp has 5 MHz quartz crystal then,

$$\text{The internal clock frequency} = \frac{5}{2} = 2.5 \text{ MHz}$$

$$\text{Time for one T-state} = \frac{1}{2.5\times10^{6}} = 0.4 \ \mu \text{ sec.}$$

5.1.1 Time delay using one register:

The flow chart in Fig. 5.1 shows a time delay loop. A count is loaded in a register, and the loop is executed until the count reaches zero. The set of instructions necessary to set up the loop is also given Fig 5.1



.

Program:

Memory address	Opcode	Labels	Mnemonics	Operands	Comment
FC 00	06		MVI	B,10H	Get 10 in register B
FC 01	10				
FC 02	05	Loop	DCR	B	Decrement register B
FC 03	C_2		JNZ	Loop	Has the content of B become zero? No jump to loop, yes produced ahead
FC 04	02				
FC 05	FC				
FC 06	C9		RET		

Fig.5.1

To generate very small delay only one register can be used. In the above program register B has been loaded by 10H(16 decimal). Then the register B is decremented and program moves in a loop till the content of register B becomes zero. After this program returns to the main program.

Delay time calculations:

To calculate delay time it is examined that how many times each instruction of the above program is executed. Number of states required for the execution of each instruction are:

Instructions	States
MVI B,10	7
DCR B	4
JNZ	7/10
RET	10

The instruction JNZ takes 10 states when the content of register B is not zero and the program jumps to the label loop. JNZ takes only 7 takes when the content of register B has become zero and the program proceeds further to execute RET instruction. The Instructions MVI B,10 H and RET are executed only once. The instruction DCR B is executed 16 times. The instruction JNZ is executed 16 times out of which 15 times the program jumps to the label loop as the content of register B has not become zero , and takes 10 states each time. At last when the content of register B becomes zero, JNZ is executed and the program proceeds further. The last execution of instruction JNZ takes only 7 states. The number of states required for the execution of each instruction and now many times each instruction has been executed are as follows:

Instruction	How many times the instruction is executed	States
MVI B, 10 H	1	7 x 1
DCR B	16	4 x 16
JNZ	16	10 x 15+ 7 x 1
RET	1	10 x 1

Total states $= 7 \times 1 + 4 \times 16 + (10 \times 15 + 7 \times 1) + 10 \times 1$

$$= 7 + 64 + 150 + 7 + 10 = 238$$

Time for one T state for Intel 8085 is 320 ns.

Delay time $= 238 \times 320 \times 10^{-9}\ sec$

$$= 0.238 \times 0.320\, milliseconds$$

$$= 0.07616\, m\, sec.$$

To generate maximum delay register B is loaded by FF(255 decimal). The maximum delay using one register is,

$= 7 \times 1 + 4 \times 255 + (10 \times 254 + 7 \times 1) + 10 \times 1.$

$= 7 + 1020 + 2540 + 7 + 10.$

$= 3584 \times 320 \times 10^{-9}\, sec.$

$= 1.11688\, milli\, sec.$

5.1.2 Time delay using a register pair:

The time delay can be considerably increased by setting a loop and using a register pair with a 16 bit number (maximum FFFF H). The 16- bit number is decrement by using the instruction DCX. However the instruction DCX does not set the zero flag and without the test flags, Jump instructions cannot check desired data conditions. Additional techniques, therefore must be used to set the zero flag.

The following set of instructions uses a register pair to set up a time delay.

Label	Mnemonics	Operands	Comments	T states
	LXI	D, FFFF	Get FFFF in register pair DE	10
Loop	DCX	D	Decrement count	6
	MOV	A,D	Move content of register D to accumulator	4
	ORA	E	Check if D and E are zero	4
	JNZ	Loop	If D-E is not zero, jump to loop	10/7
	RET		Return to main program	10

If the count in register pair D-E is N, the total number of states are:

$\text{States} = 10 + N(6 + 4 + 4) + (N - 1) \times 10 + 1 \times 7 + 10$

$= 24N + 17.$

$\text{Delay} = (24N + 17) \times \text{time for one state}$

Maximum delay will be occur when count N=FFFF

$= 65{,}535 \text{ decimal}$

$\text{Maximum delay} = (24 \times 65{,}535 + 17) \times 320 \times 10^{-9} sec$

$= 20.97664 \text{ milliseconds}$

5.1.3 Delay subroutines using TWO register: (time delay using a loop with in a loop technique):

A time delay similar to that of a register pair can also be achieved by using two loops, one loop inside the other loop, as shown in Fig5.2(a). For example, register C is used in the inner loop (Loop 1) and register B is used for the outer (Loop 2). The following instructions can be used to implement the flow chart shown in Fig 5.2(a).

	MVI B,10H	Get desired number in register B
Loop 1	MVI C, 78H	Get desired number in register C
Loop 2	DCR C	Decrement C
JNZ	Loop 2	Is [c] zero? No , go to loop 2. yes, produced further.

DCR	B	Decrement register B
JNZ	Loop1	Is [B] is zero? No, go to loop 1. Yes ,produced further.
RET		Return to main program

Delay calculations:

States for instructions used are:

Instructions	States
MVI	7
DCR	4
JNZ	7/10
RET	10

The states for the execution of each instruction of the program are as follows:

Instructions	How many times executed	States
MVI B, 10H	1	7x1
Loop 1 MVI C, 78H	16	7x 16
Loop2 DCR C	120x 16	4x 120 x 16
JNZ Loop 2	120 x 16	10x[(120-1)x16]+7x16
DCR B	16	4x 16
JNZ Loop 1	16	10x(16-1)+7x1
RET	1	1x10

Total states: $= 7 \times 1 + 7 \times 16 + 4 \times 120 \times 16 + 10 \times (120 - 1) + 7 \times 16 + 4 \times 16 + 10 \times (16 - 1) + 7 \times 1 + 1 \times 10$

$= 27182$ states.

Delay time $= 27182 \times 320 \times 10^{-9}$ second$= 8.6912$ milliseconds.

To get maximum delay using two registers both register B and C are loaded by FF. The total number of states for the maximum delay is,

$= 7 \times 1 + 7 \times 255 + 4 \times 255 \times 255 + 10(255 - 1) \times 255 + 7 \times 255 + 4 \times 255 + 10 \times (255 - 1) + 7 \times 1 + 10 \times 1.$

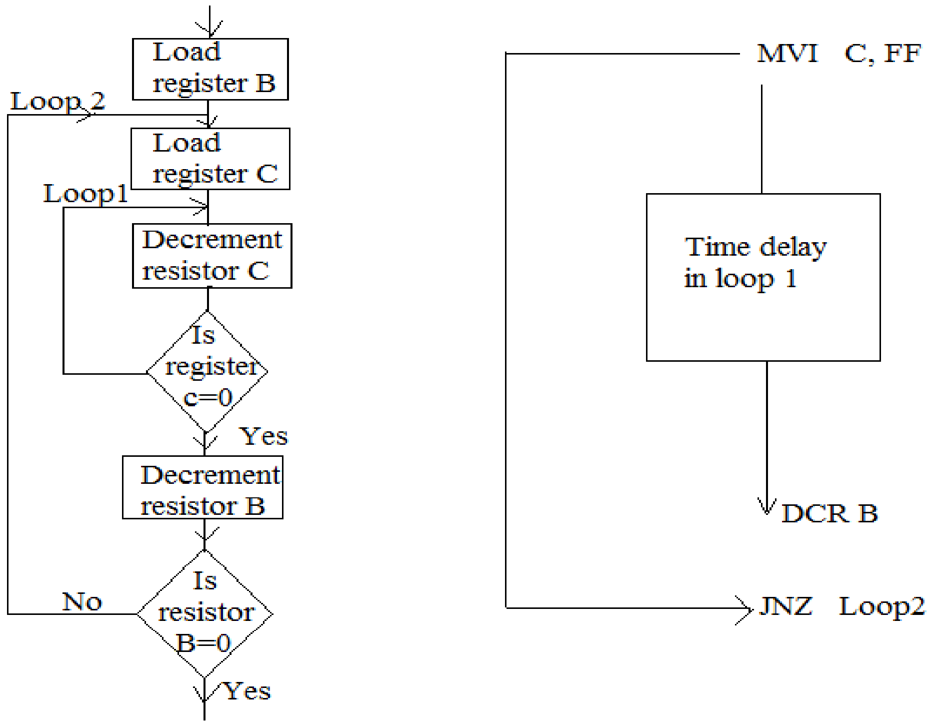
$= 7 + 1785 + 260100 + 647700 + 1785 + 1020 + 2540 + 7 + 1.$

$= 914954$ states

Delay time $= 914954 \times 320 \times 10^{-9}$ seconds

≈ 0.293 seconds.

Thus we see that delay obtained is very small. For some purpose this time is sufficient. If more delay is required 3 or more registers can be used.



(a) (b)

Fig.5.2

Similarly , the time delay with in a loop can be increased by using instructions such that will not affect the program except to increase the time delay. For example, the instruction NOP (no operation) can add four T states in the delay loop. The desired time delay can obtained by using any or all available registers.

5.1.4 Delay subroutine using 3 register:

The delay program using 3 registers is given below. To see the indication after certain delay one LED display can be incorporated. The interfacing circuit of a simple LED is shown in Fig.5.3. The LED display is connected to the pin PB_0 of the port B through the buffer. A 560 ohm register is used in series with the LED to limit the circuit drawn by it. A pull up register of about 1 k is used to boost the output voltage of the buffer. The control wordis 98 H, which makes port B as output port.

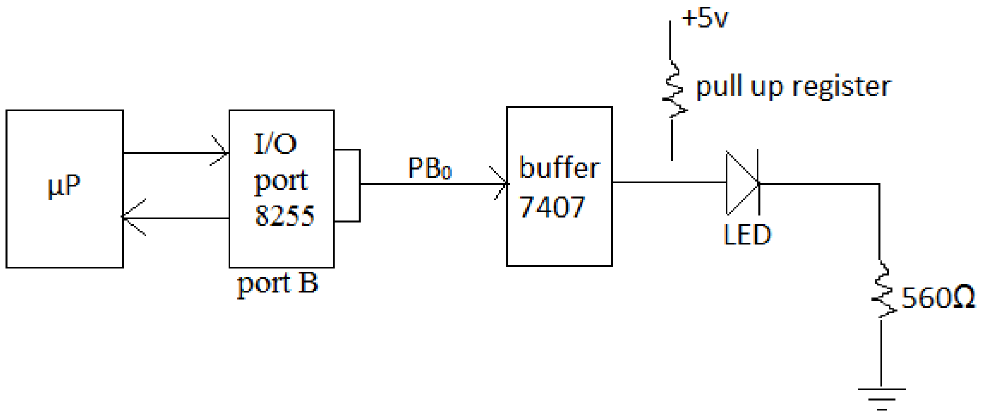


Fig 5.3: Interfacing of LED display

Program:

Memory address	Machine codes	Labels	Mnemonics	Operands	Commends
2400	3E		MVI	A, 98 H	Get controlled word.
2401	98				
2402	D3		Out	03	Initialize port for LED display
2403	03				
2404	06		MVI	B, 50 H	
2405	50				
2406	0E	Loop 1	MVI	0, FF	
2407	FF				
2408	16	Loop 2	MVI	D,FF	
2409	FF				
240A	15	Loop 3	DCR	D	
240B	C2		JNZ	Loop 3	
240C	OA				Delay subroutine with 3 register
240D	24				
240E	OD		DCR	C	
240F	C2		JNZ	Loop 2	
2410	08				
2411	24				
2412	05		DCR	B	
2413	C2		JNZ	Loop 1	
2414	06				

2415	24				
2416	3E		MVI	A, 01	
2417	01				
2418	D3		Out	01	Output for LED.
2419	01				
241A	76		HLT		

The numbers 50 in register B, FF in C and FF in D are moved to get the desired delay. The delay time with to get the desired delay. The delay time with these numbers is a about 25 sec. This time has been noted by stop watch in the laboratory. The maximum delay with 3 registers can be obtained when all registers are loaded with FF. The maximum delay is about 74 sec.

Example 1:

Write a delay routine to produce a time delay of 0.5 m sec in 8085 µp based system whose clock source is 6 MHz quartz crystal.

Solution:

The delay required in 0.5 m sec, hence an 8- bit register of 8085 can be used to store a count value and then decrement to zero. The delay routine is written as a subroutine as shown below:

Loop	MVI	D,N	Load the count value, N in D register.
	DCR	D	Decrement the count
	JNZ	Loop	If the count is not zero go to the loop
	RET		Return to main program

The following table shown the T-state required for execution of the instruction in the subroutine.

Instruction	How many times the instruction is executed	States
MVI D,N	1	7
DCR D	N times	4 xN=4N
JNZ loop	(N-1) times	10x(N-1) (or)7x1 =7
RET	1	10 x 1

Total T-state required for subroutine:

$$= 7 + 4N + 10(N - 1) + 7 + 10.$$

$$= 7 + 4N + 10N - 10 + 7 + 10.$$

$$= 14N + 14.$$

<u>Calculation to find the count value N</u>

External clock frequency =6MHz

Internal clock frequency $= \frac{\text{Externalclock}}{2} = \frac{6}{2} = 3MHz$

Time period of one T-state = $\frac{1}{\text{internal clock frequency}}$

$$= \frac{1}{3 \times 10^6} = 0.3333\ \mu\ sec$$

$$\approx 330\ n\,\text{sec}$$

Number of T-states required for 0.5 m sec

$$= \frac{Required\ time\ delays}{Time\ for\ one\ states.}$$

$$= \frac{0.5 \times 10^{-3}}{0.3333 \times 10^{-9}}$$

$$= 1500.15 = 1500_{10}$$

∴ Number of T- states required for 0.5 m sec $=1500_{10}$

On equating the total T-states required for subroutine and number of T- states for the required time delay the count value, N can be calculated.

$$\therefore 14N + 14 = 1500_{11}$$

$$N = \frac{1500 - 14}{14}$$

$$= 106.14 \approx 106_{10} \approx 6A_H$$

∴ Count value N=$6A_H$

If the above delay routine is called by a program and executed with value of $6A_H$ then the delay produced will be 0.5 m sec.

5.2 Generation of square waves or pulses using μp:

A square wave or pulse can easily be generated by microprocessor. The μp sends high and then low signals to generate square wave or pulse. A pulse or square wave can be generated using I/O port or SOD line or timer/ counter (Intel 8253)

To generate square wave connections are made in Fig. 5.4. The pin PB_0 of the port B of 8255-2 is used for taking output. This is connected to a buffer 7407. The final output is taken from the buffer terminal. The control word used in the program is 98H to make port B an output port.

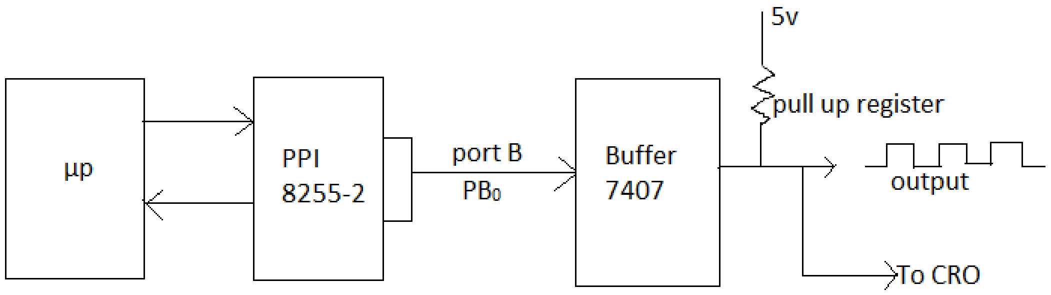


Fig 5.4: To generate square wave using μp

Program:

Address	Machine codes	Labels	Mnemonics	Operands	Commands
2400	3E		MVI	A,98H	Get control word
2401	98				
2402	D3		OUT	OB	Initialize ports
2403	OB				
2404	3E	Loop	MVI	A,00	
2405	00				
2406	D3		OUT	09	Make PB_0 low
2407	09				
2408	CD		CALL	Delay 1	
2409	00				
240A	25				
240B	E		MVI	A, 01	
240C	01				
240D	D3		OUT	09	Make PB_0

					High
240E	09				
240F	CD		CALL	DELAY 2	
2410	09				
2411	25				
2412	C3		JMP	Loop	

Subroutines:

Delay 1:

2500	06		MVI	B,02	Get count for delay
2501	02				
2502	05	G_0	DCR	B	
2503	C3		JNZ	G_0	
2504	02				
2505	25				
2506	C9		RET		

Delay 2:

2509	OE		MVI	C,02	Get count for delay
250A	02				
250B	0D	Back	DCR	C	
250C	C2		JNZ	Back	
250D	0B				
250E	25				
250F	C9		RET		

In this program Delay 1 controls the time period for which the square wave remains low, ie, zero. Delay 2 controls the time for which the wave remains High ie, 1. If the time period for Low and High are to be kept equal the counts in register B and register c are made equal. For such a case there is no need of two subroutines only one delay subroutine will be called at two places ie, at 2408 and 240F memory addresses .There will be slight difference in timing of Low and High due to the instruction JMP loop. If accuracy is desired this can be adjusted by suitable adjustment in the cunts of register B and register C. The difference can also be minimized by inserting two NOP instructions in Delay 1 subroutines. The instruction JMP Loop has been used at the end of the program to repeat the whole process to generate equal wave.

5.2.1 To generate square wave or pulse using SOD line:

A square wave can also be generated using SOD line of the µp. zero and one can be outputted at SOD lines using SIM instruction. The execution of SIM instruction loads the content of the 7th bit of the accumulator into SOD latch. While executing SIM instruction the 6th bit of the accumulator is set to 1 to enable SOD lines. To generate square wave the connections are made as shown in Fig 5.5. The SOD terminal is available on a µp kit.

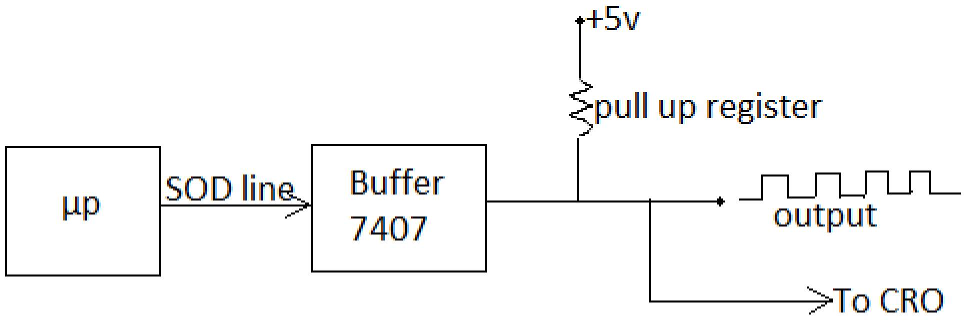


Fig 5.5: To generate square wave using SOD lines

Program:

Memory address	Machine codes	Labels	Mnemonics	Operands	Comments
2400	3E	Loop	MVI	A, 40H	6th bit of accumulator 1 and 7th bit zero.
2401	40				
2402	30		SIM		Make SOD line low
2403	CD		CALL	DELAY 1	
2404	00				
2405	25				
2406	3E		MVI	A,CO	6th bit of accumulator 1 and 7th bit 1
2407	C0				
2408	30		SIM		Make SOD line High
2409	CD		Call	Delay 2	
240A	09				
240B	25				
240C	C3		JMP	Loop	
240D	00				
240E	24				

Delay 1 and Delay 2 are subroutines to control time periods for which SOD line remains Low and High respectively. To output Low on the SOD line the 7th bit of the accumulator is set to zero. The 6th bit is set to 1 as it is for enabling the SOD line. Other bits of the accumulator are for setting/resetting of Interrupts. For this case other bits are zero or one, it is immaterial. In the above program they have been set to zero. The first instruction of the program MVI A, 40 indicates that the 7th bit is zero, 6th bit is one and other bits are zero. The SIM instruction outputs the 7th bit of the accumulator on SOD line. CALL Delay 1 is for controlling the time period for which the square wave remain low. The instruction MVI A, CO indicates that the 7th bit of the accumulator is one, 6th bit one and other bits zero. Now the execution of SIM instruction outputs High, ie, 1 on the SOD line. Call Delay 2controls the time period for which the square wave remains High. The JMP loop instruction repeats the whole process to generate a square wave.

5.3 Sensors and Transducers:

5.3.1 Transducers or Transductors:

A general term used for any device receiving input power from one system and supplying output power corresponding to the input certain characteristics(e.g wave from) to another system; which may electrical, mechanical ,or acoustic and thus including transformers, amplifiers, filters, microphones , loud speakers etc.,

Transducers syn.sensors:

Any device that converts a non-electrical parameters ,e.g. sound , pressure, or light, into electrical signals or vice versa. The variations in the electrical signal parameters are functions of the input parameter. Transducers are used in the electro acoustic field. Gramophone pickups, microphones, and loud-speakers are all electro acoustic transducers. The term is also applied to device in which both the input and output electrical signals. Such a device is known as an electric transducers.

For the measurement of physical quantities like temperature, pressure, speed, flow etc., transducers are used to current them to electrical quantities. The electrical output of the transducers is proportional to the input quantity which may be temperature or speed or any other physical quantity.

A transducers used for strain measurement is called a strain Gauge. Strain gauges are used to measure strains and stresses in structures and strain.

Sensors:

A transducers or device whose input is a physical phenomenon and whose output is a quantitative measure of that physical phenomenon.

5.4 Measurement of physical quantities:

Microprocessor – based systems are widely used in industries for the measurement, display and monitoring of physical quantities like temperature, pressure , speed , flow etc., Transducers are used to convert the physical quantities into electrical signal. If the electrical signal is small it is amplified ,using amplifiers. The electrical signal is applied to an A/D converter which is connected to a µc. It more than one physical quantities are to be monitored a multiplexer is included in the interface. A schematic diagram for general interface is shown in Fig.5.6

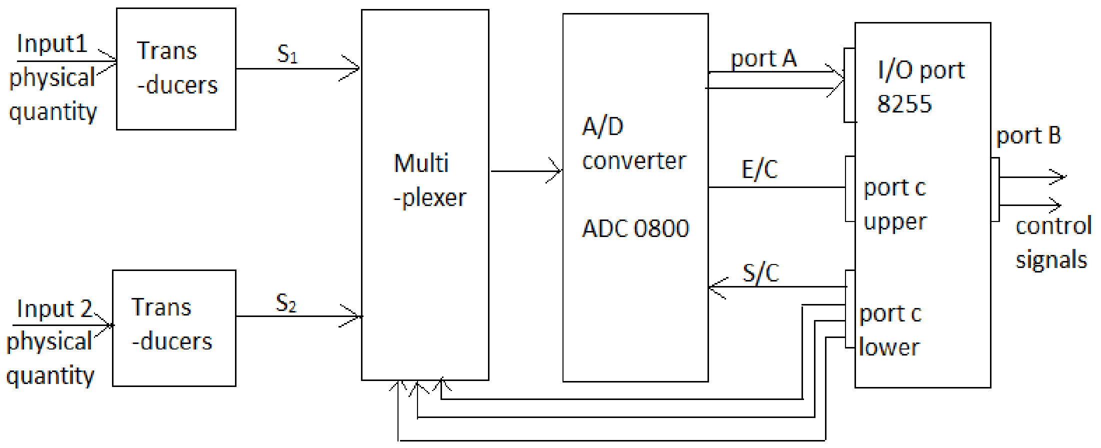


Fig 5.6: Schematic diagram of Interface for physical quantity measurement

5.4.1 Temperature measurement and control:

For the measurements of temperature one of the following devices are used.

(i) Resistance thermometers (-100 to +300^{0}c)

(ii) Thermo couples (-250 to +200^{0}c)

(iii) Thermistor (-100 to +100^{0}c)

(iv) Pyrometers(+100 t0 +5000^{0}c)

(i) Resistance thermometers:

Platinum wires are frequently used in resistance thermometers for industrial applications because of greater resolution, and mechanical and electrical stability as compared to copper or nickel wires.

A change in temperature causes a change in resistance. The resistance thermometer is placed in an arm of a wheatstone bridge to get a voltage proportional to temperature.

(ii) Thermistor:

A thermistor is a semiconductor device fabricated from a sintered mixture of metal alloys having a large negative temperature co efficient. A thermistor is used in a wheatstone bridge to get a voltage proportional to temperature. It can be used in the range of -100^{0}c to +100^{0}c for greater accuracy as compared to platinum resistance thermometer.

(iii) Thermocouple:

A two-terminal device based on the seeback effect, which is composed of two dissimilar metals that produce a voltage across function that is linearly proportional to the temperature nof the junction.

A thermocouple is the most widely used transducer to measure temperature. Thermocouple materials for the different range of temperatures are as follows:

Material	Temp . range ^{0}c
Iron-constantan	-200 to +1300
Chromel – alnmel	-200 to +1200
Copper – constantan	-200 to +400
Pt – Rh - Platinum	0 to 1500
Tungsten – rhenium	O to 2000

5.4.2 Microprocessor – Based scheme:

Fig 5.7(a) shows a microprocessor-based scheme for temperature measurement and control. The output of a thermocouple proportional to the furnace or oven etc,. is in millivolt. It is to be amplified using multistage amplifier before it is processed by microprocessor. The amplified voltage is applied to an A/D converter. The µp sends a start of conversion signal to the A/D converter through the port of 8255PPI. When A/D converter completes conversion, it sends an end of conversional signal to microprocessor. Having received an end of conversion signal from A/D converter the microprocessor reads the output of the A/D converter which is a digital quantity proportional to the temperature to be measured. The µp displays the measured temperature. If the temperature of a furnace, oven or water- bath is to be controlled, the µp first measures its temperature and then compares the measured temperature with a reference temperature at which the temperature is to be maintained. If the measured temperature is higher than the reference temperature, µp sends control signal to reduce temperature .If the measured temperature is less than the reference temperature, the µp sends a control signal to increase temperature. The temperature of a furnace or oven can be increased or decreased by increasing or decreasing the final input to the furnace. If heating is done by electric heaters, current in heating element is controlled.

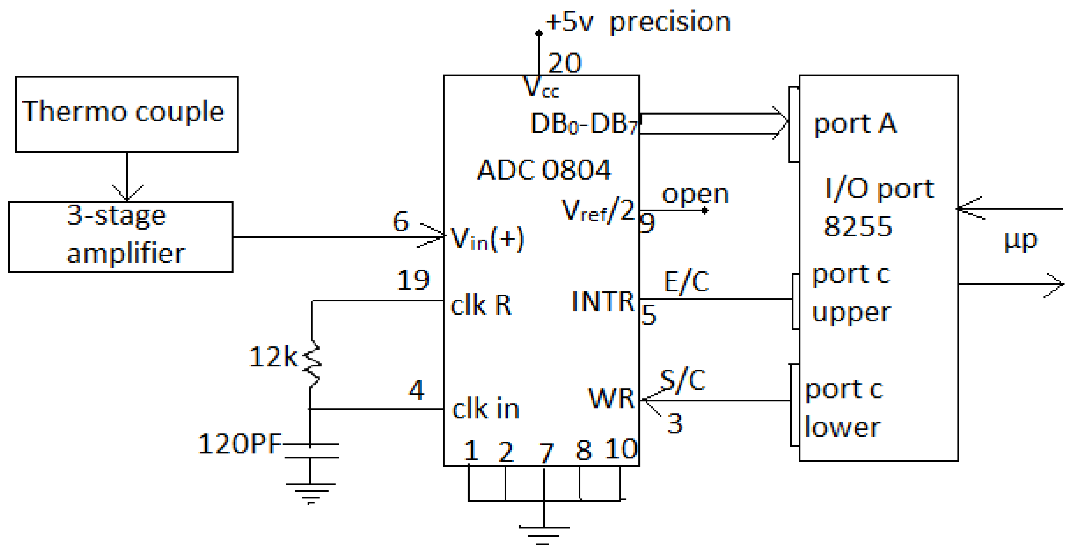


Fig 5.7(a): µp based scheme for temperature measurement

Fig. 5. 7(b) shows an amplifier circuit to amplifier the output of the thermocouple, D.C. level indicator is for initial adjustment.

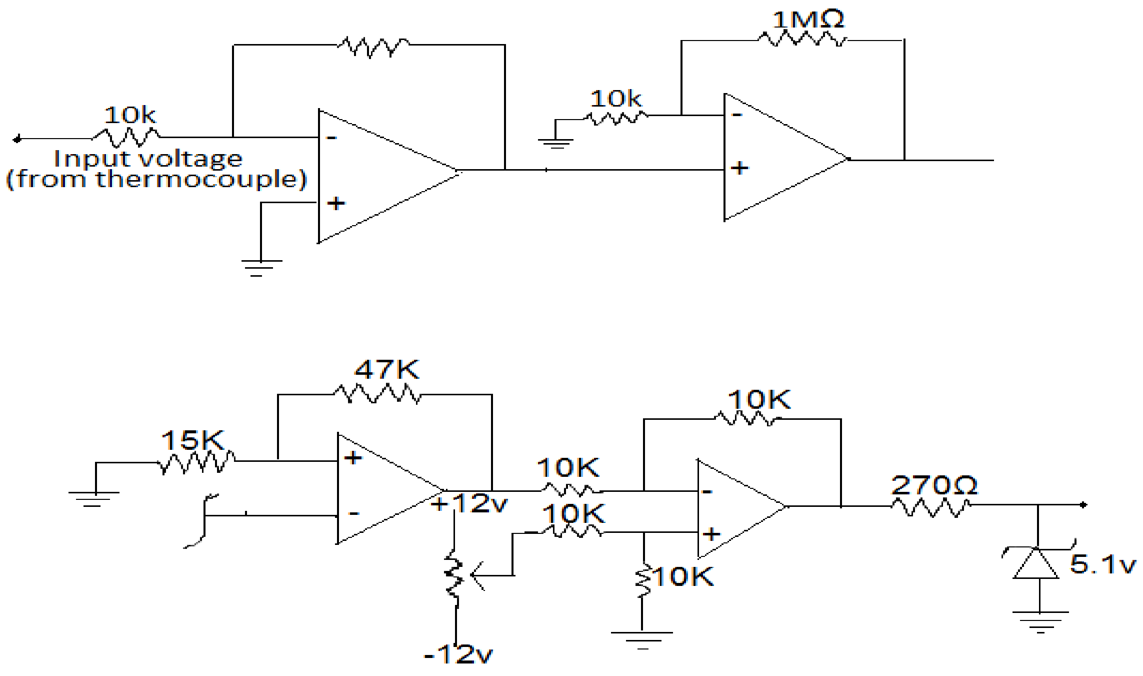


Fig 5.7(b): Three stage Amplifier and D.C. Level Detector

Program:

Machine address	Machine codes	Label	Mnemonics	Operands	Comments
FC00	3E		MVI	A,98	Control word for 8255
FC01	98				
FC02	D3		OUT	0B	Controlled register address
FC03	0B				
FC04	3E	Loop	MVI	A,00	
FC05	00				
FC06	D3		OUT	0A	
FC07	OA				
FC08	3E		MVI	A,08	
FC09	08				
FC0A	D3		OUT	0A	
FC0B	0A				
FC0C	DB	READ	IN	0A	
FC0C	0A				
FC0E	17		RAL		Check end of conversion
FCOF	DA		JC	READ	
FC11	OC				
FC12	DB		IN	08	Read output of A/D converter
FC13	08				
FC14	32		STA	FFF6	Store result
FC15	F6				
FC16	FF				
FC17	06		MVI	B,00	Display result
FC18	00				
FC19	CD		CALL	FA06	
FC1A	06				
FC1B	FA				
FC1C	C3		JMP	Loop	
FC1D	04				
FC1E	FC				

Result:

Temperature in ^{0}c	Transducer voltage in mV	Amplified voltage volts	Digital voltage
45	0.3	0.66	2D
50	0.5	0.95	38
55	0.8	1.29	44
60	0.9	1.56	51
65	1.2	2.03	62
70	1.4	2.19	70
75	1.5	2.49	7F
80	1.7	2.69	8B
85	2.0	3.05	9E
90	2.2	3.33	AA
95	2.5	3.72	C1
100	2.7	3.99	CA

5.4.3 Temperature Monitoring System:

Two transducers have been shown in figure 6. These transducers sense the temperature of two ovens. The temperatures of these two ovens are to be maintained at T_1 and T_2 respectively. The µp sends command to switch on the channel s_1 to get the electrical signal proportional to the temperature of oven no 1. Then it sends start of conversion pulse S/C to the A/D converter. After the conversion is over the A/D converter sends end of conversion signal E/C to the µp. on receiving E/C signal the µp reads the output of the A/D converter. The output of the A/D converter is a digital voltage. This is proportional to t_1, the temperature of the oven no 1. The µp compares t_1 with T_1. If t_1 is less than T_1, the µp issues a control signal to raise the temperature of the oven no 1. If the heating of the oven is electrical and current is controlled by thyristors , the µp will directly control the firing circuit of thyristors to increase current resulting in increase of the temperature. If the heating is done by final, the µp will send a signal to the relay which is controlling the final input. The measurement temperature is also displayed. If $t_1>T_1$, the µp sends signal to decrease it till it becomes equal to T_1. After this the µp sends commands to switch on the multiplexer channel S_2 to get the electrical voltage proportional to the temperature of over no 2. The µp measures and control its temperature as explained above. After certain internal of time the µp repeats the process of measuring and controlling the temperature of the two ovens. A program flow chart is shown in Fig. 5.8. Similarly any other physical quantity can be measured and monitoring continuously. If a transducers given A.C signal it can be rectified using precision rectifiers.

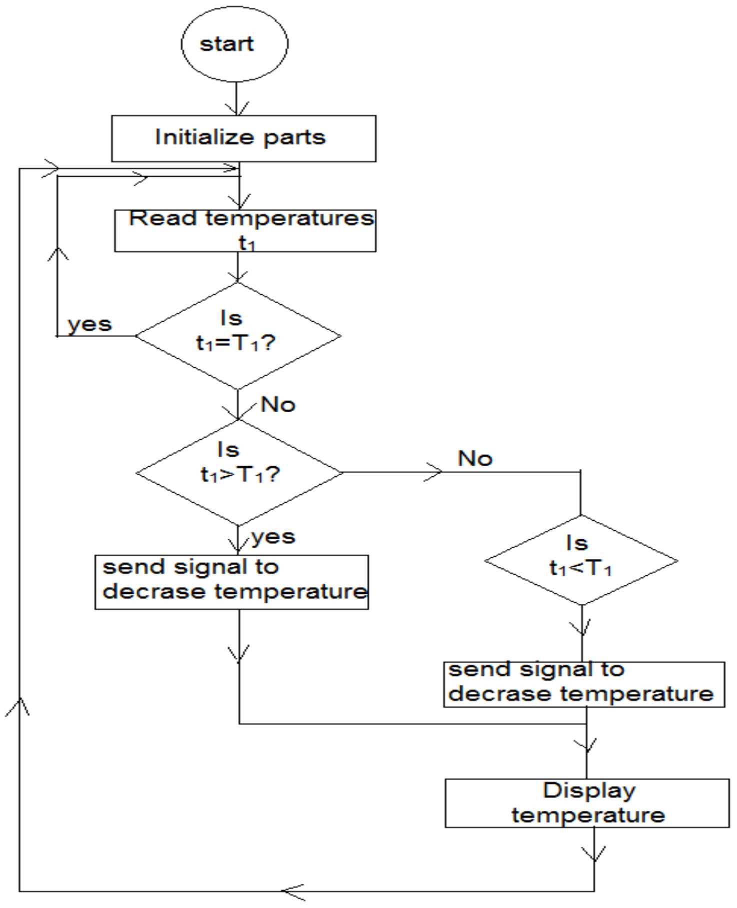


Fig 5.8: Program flow chart for temperature display and monitoring

5.5 Measurements of Electrical quantities:

(i) Frequency measurements:

To measure the frequency of a signal, the time period for half cycle is measured which is inversely proportional to the frequency. A sinusoidal signal is converted to square wave using a voltage comparator LM 311 or operational amplifier LM 747 or LM 324 as shown in Fig.5.9.

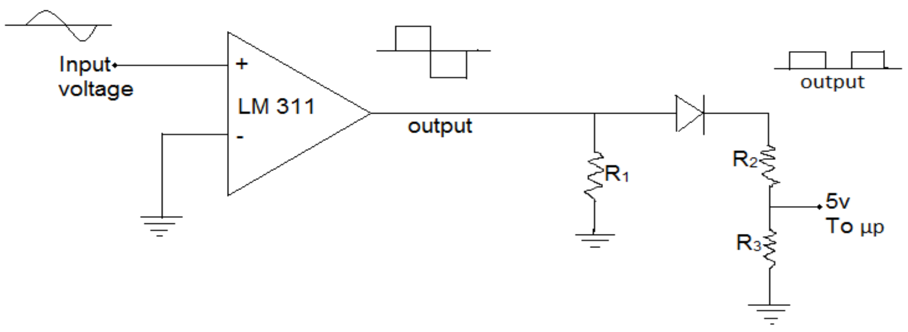


Fig 5.9: Sine to square wave generator

A diode is used to rectify the output signal. A potential divides is used to reduce the magnitude to 5 volts.

A program has been developed to sense the zero instant of the rectified square wave. The µp measures the magnitude of the square wave at two consecutive points as shown in fig 10. The two magnitudes are compared and decision is taken on the basis of carry and zero states flags, whether the point is at zero point.

Various points have been shown in Fig 5.10. Very nearly to P_3 at its left side the magnitude of the square wave is zero, and at P_4 , 5v, at logic '1'. The µp subtracts the 1st value from the 2nd, so the result is non- zero and there is no carry. This is the basis for the selection of zero instant point. Suppose the µp takes reading at p_1 and p_2 where both magnitudes are zero. Difference of the two is zero. So this is not the zero instant of the wave. At points p_5 and p_6 the difference of the two values is zero. So it is also not a zero instant point. At p_7 and p_8, the difference is non-zero but that is carry. So it is the end point of the half- square wave.

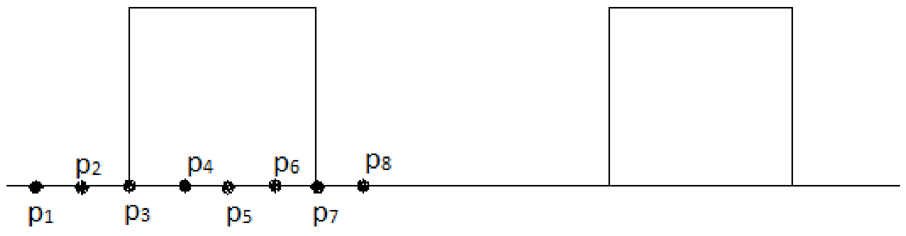


Fig 5.10: Rectified square wave

As soon as the zero instant point is detected the µp initiates a register pair to count the number how many times the loop is executed. The µp reads the magnitude of the square wave again and moves in the loop. It crosses the loop when the magnitude of the square wave becomes zero. Thus the time for half cycle is measured. The count can be compared with the stored numbers in the look-up table and the frequency can be displayed. The count is inversely proportional to the frequency of the input signal can be used for further processing and control as desired. An interfacing circuitary is shown in Fig 5.11. The program flow chart is shown in Fig 5.12. The port is input control word is 98H.

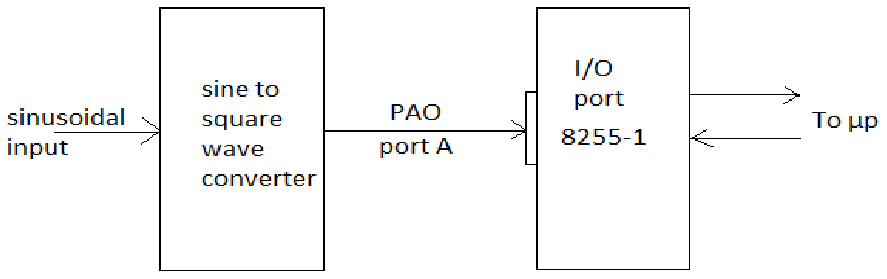
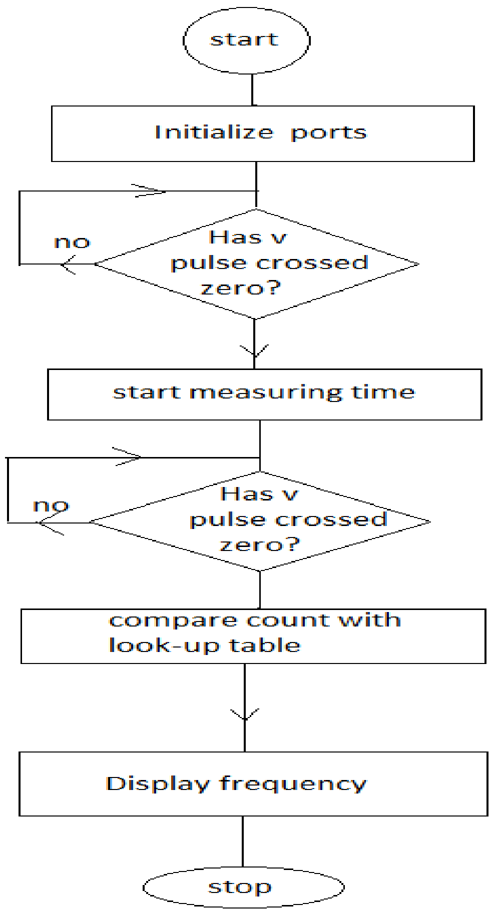


Fig 5.11: Interface for frequency measurement

Flow chart for frequency measurements:



start
Initialize ports
no
Has v pulse crossed zero?
start measuring time
no
Has v pulse crossed zero?
compare count with look-up table
Display frequency
stop

Program:

Memory Address	Machine codes	Label	Mnemonics	Operands	Commands
2000	3E		MVI	A,98H	Get control word
2001	98H				
2002	D3		OUT	03	Initialize the ports
2003	03				
2004	DB	Back	In	00	Read voltage pulse at port A
2005	00				
2006	47		MOV	B,A	
2007	DB		IN	00	Read voltage pulse again at port A
2008	00				

2009	B8		CMP	B	Compare two readings
200A	CA		JZ	Back	
200B	04				
200C	20				
200D	DA		JC	Back	
200E	04				
200F	20				
2010	01		LXI	B, 00, 00	Initialize B-C pair for counting
2011	00				
2012	00				
2013	03	Loop	INX	B	
2014	DB		IN	00	Read voltage pulse
2015	00				
2016	1F		RAR		
2017	DA		JC	Loop	Check whether v has become zero, no, go to loop
2018	B				
2019	20				
201A	76		HCT		stop

Fig.5.12

In the above program, the part of the program from the memory address 200 4 to 200 F is to detect the zero instant of the square wave. In 00 at the memory location 200 4 is the 1st reading of the magnitude of the square wave. MOV B,A transfers the 1st reading from the accumulator to the register B. Again In 00 at the memory address 200 7 takes the 2nd readings of the magnitude CMP B compares these two readings. JZ indicates the condition that both readings are equal magnitudes, ie. Readings are either 0,0 or 1,1. Therefore it is not the condition of zero constant and the program jumps to the label back. If indicates that the 1st readings is 1 and the 2nd reading 0. This is the condition for the end of rectifies square wave. So program again jumps to the label back. The program will move further only when the result is non-zero and there is no carry. This will be the case when the 1st reading is 0 and the 2nd readings is 1/ this is the case of zero instant. Once the zero instant is detected the program moves loop till the square wave exists. Thus the time period for half cycle is measured. The count is inversely proportional to the frequency . If frequency is to be displayed 7- segment displays can be interfaced and using loop-up table technique it can be displayed.

(ii) Frequency measurement using SID line:

The sinusoidal wave is converted into square wave using op.amp. The output is reduced to 5V and applied to SID line of a µp. The execution of RIM instruction reads the status of SID line and stores it in the 7th bit of the accumulator. It also reads status of interrupt MASKS and STORES them in other bits of the accumulator. At present interest is in the status of SID line, ie contant of the 7th bit of the accumulator. A program to measure frequency is given below. Program from the memory location 2400 to 240D detects the rising edge of the square wave. Once it is detected H-L pair is initialized for counting. The program checks how long the wave remain high in one cycle. In other words it measures time for half cycle because the wave remains high only for half cycles; for other half cycle it goes low. The time is given interms of counts which is proportional to the wave frequency of the wave. The count is shared H-L pair. Frequency can be displayed using look-up table or can be computed and then displayed. The counts in H-L pair can be used for control which depends on frequency.

Program:

Memory address	Machine codes	Label	Mnemonics	Operands	Comments
2400	20	Loop	RIM		Read frequency signal
2401	E6		ANI	80	7th bit of the accumulator is retained.
2402	80				
2403	4F		MOV	C,A	
2404	20		RIM		Read frequency signal again
2405	E6		ANI	80	
2406	80				
2407	B9		CMP	C	Compare with previous value
2408	CA		JZ	Loop	Detect low to high transition
2409	00				
240A	24				
240B	DA		JC	Loop	
240C					
240E	21		LXI	H, 0000	Initiative H-L pair to count

					frequency
240F	00				
2410	00				
2411	23	Count	INX	H	Count frequency
2412	20		RIM		Read frequency of signal
2413	17		RAC		
2414	DA		JC	Count	Is frequency signal high yes, to count
2415	11				
2416	24				
2417	76	HLT			

Result:

Frequency	Content of H-L pair(count for half cycle)
50	026A
60	0203
100	0135
120	0102
200	0094

(iii) Resistance Measurement:

The resistance of a circuit is given by the expression

$$R = Z\cos\varphi$$

$$= \frac{V_{rms}}{I_{rms}}\cos\varphi$$

$$= \frac{K_1 V_m \cos\varphi}{K_2\, I_{dc}}$$

$$= K\,\frac{V_m\ \cos\varphi}{I_{dc}}$$

Where K_1 , K_2 and K are constants.

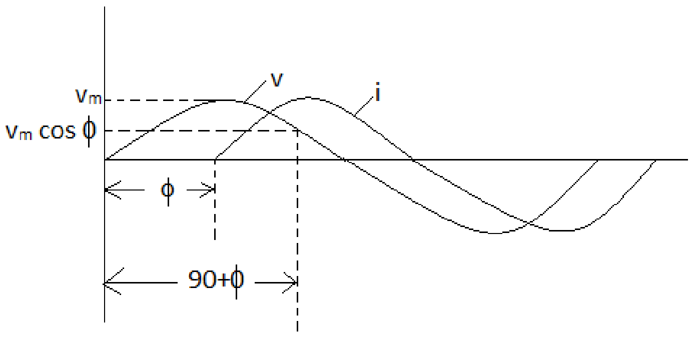


Fig 5.13: Instantaneous values of voltage at the instant of peak current

The instantaneous value of the voltage at the moment of the peak current is equal to $V_m \cos\varphi$, Fig 5.13. To obtain a pulse at the moment of peak current a phase shifting circuit and zero- cross detector have been designed and developed. The current signal is fed to the phase shifter to get a phase shift of 90^0. Thus the output of the phase shifter is fed to the zero- cross detector as shown in Fig 5.14. To obtain the required pulse. The µc reads and examines the output of the zero-cross detector whether the pulse has appeared. The appearance of the pulse indicates that the current signal has reached its peak. At this moment the µc sends a command to the multiplexer to switch on the channel S_4 to obtain the instantaneous value , which is equal to $V_m \cos\varphi$.

The instantaneous value of the voltage is fed to an analog to digital converter. After the conversion is over, the digital voltage is stored in the memory. To obtain the rectified current signal a precision rectifier using I.C. packages LM 398 has been used. The µc gets the rectified current through the multiplexes channel S_7 and A/D converter. After receiving the values of $V_m \cos\varphi$ and I_{dc} the microcomputer calculates the value of resistance.

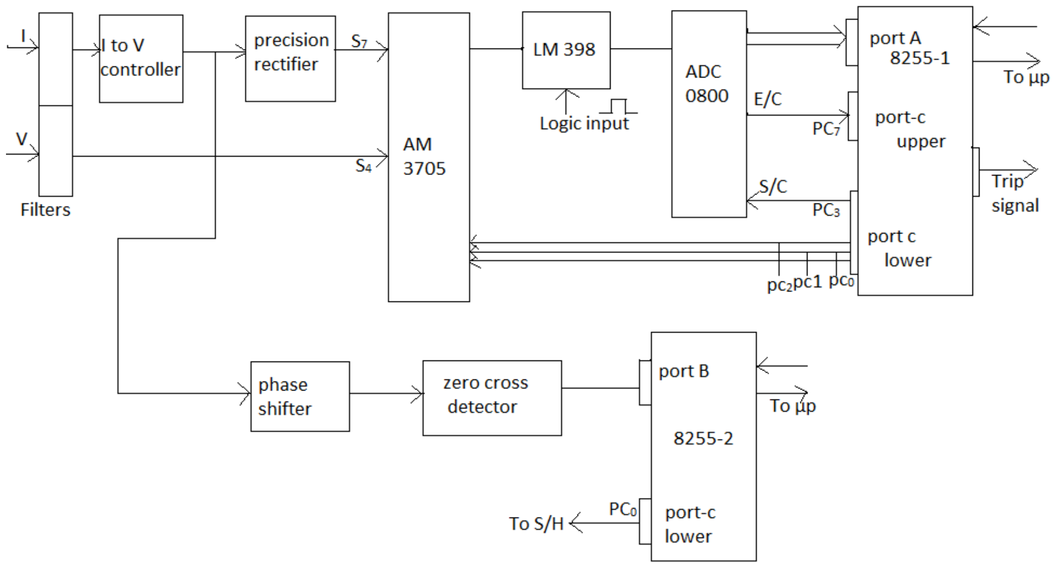
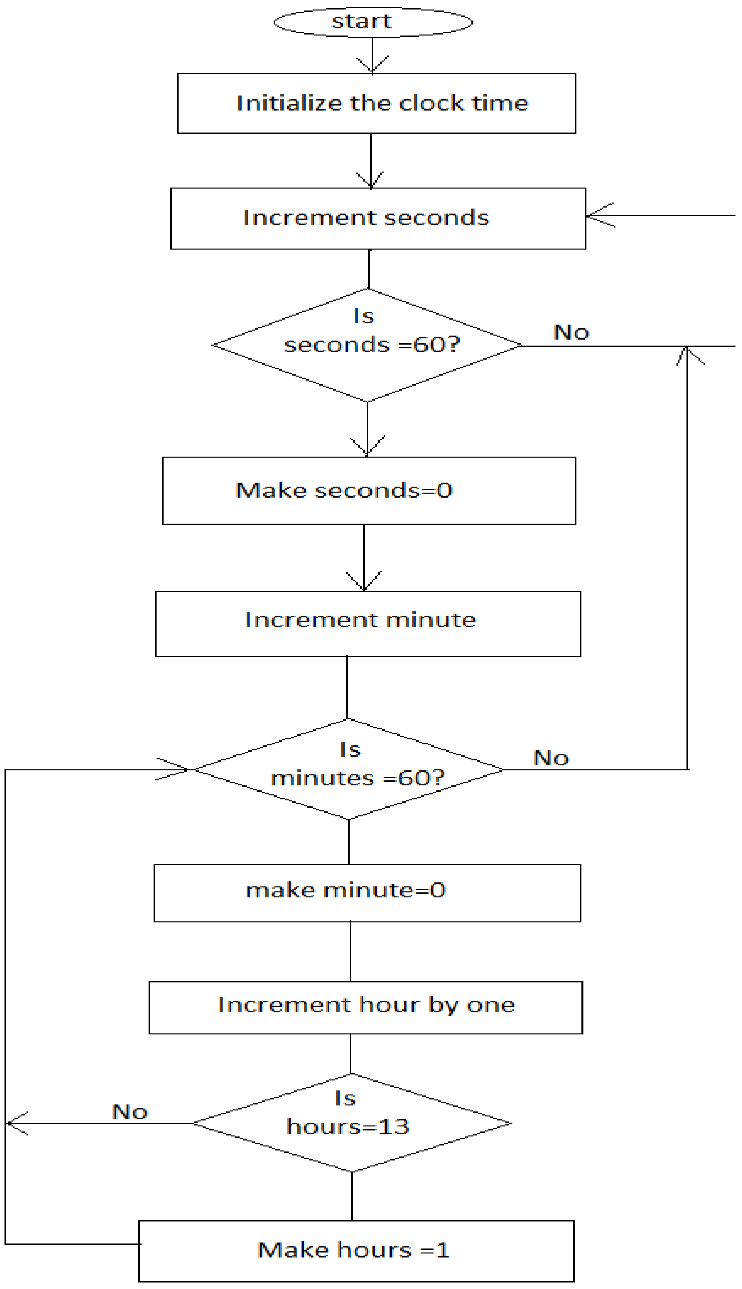


Fig 5. 14: Schematic diagram of Interface for resistance measurements

5.6 Digital clock:

Many µp applications would require doing certain tasks at specific time of the day or involve the time of day in some other form. For example, switching ON and OFF street lights at specific time in the evening and following morning or punching entry time of every worker for a shift, at a factory gate etc,. A basic requirement of this type of application is a real time digital clock with a display of current time. The flow chart and the corresponding program are shown in Fig.5.15.

Flow Chart for digital clock program:



Program:

Label	Mnemonics	Comments
BEGIN	LXI H,4003	
	MOV D,M	Initialize hours in D register
	INX H	
	MOV E,M	Initialize minute in E register
	INX H	
	MOV A,M	Initialize seconds in accumulator
	ADI 01	Increment accumulator
	DAA	Convert into BCD Signal
	MOV M,A	
	CPI 60	Compare seconds with 60
	JZ UPDATE	If seconds=60 go to UPDATE
DISP	LXI H,4005H	Initialize the with specific address
	MOV A,M	Load accumulator with display the seconds in data field of kit.
	CALL 0520	
	DCX H	
	MOV D,M	Load hours into e register
	DCX H	
	MOV D,M	Load hours into D register
	CALL 0510	Display the hours and minutes in the address field of kit

```
DELAY           LXI  B,16-bit data              Initialize BC register with a

                                                value  so as get 1 sec delay

TOP             DCX  B

                MOV A,C

                ORA  B                          Check BC register =0000

                JNZ   TOP                       If BC is not equal to 0000, go to
                                           top

                JMP   BEGIN                     Go to begin

UPDATE          SUB  A                          Initialize A register with 00

                MOV  M,A

                MOV A,E

                ADI  01                         Increment minutes
           DAA                             Convert into BCD format

                STA  4004                       Store minutes into specific address

                CPI  60                         Compare minutes =60

                JNZ  DISP                       If minutes is not equal to 60, go to
                                           disp

                SUB  A                          Initialize ACC=00

                STA  4004 H                     Store 00 into minute

                MOV A,D

                ADI  01                         Increment hours

                DAA                             Convert into BCD format

                STA  4003 H                     Stores hours into specific address.

                CPI  B                          Check hours =B

                JNZ   DISP                      If hours is less than B, go to DISP

                MVI  A,01                       Load ACC=01
```

STA 4003H	Store 01 into hours
JMP DISP	Go to disp.

Fig. 5.15

Procedure:

1. Load the given program starting from the specified memory location or from any other suitable memory location.
2. Initialize the specified or suitable location with the starting values of hours, minutes and seconds
3. Use the relevant display routine of your trainer kit displaying data at the address and data field.
4. Run the program.

5.7 DC Motor Speed Control:

A µp trainer kit is used in thus DC motor speed control system. The interface circuit along with the trainer kit is designed to maintain the speed of a DC motor at some specified speed, to display the measured speed on the LED displays and to set the speed using keyboard.

(i) Hardware Design:

As the µp trainer kit is used in this system, the hardware design deals only the interface circuit. The block diagram for extra hardware as shown in Fig.5.16

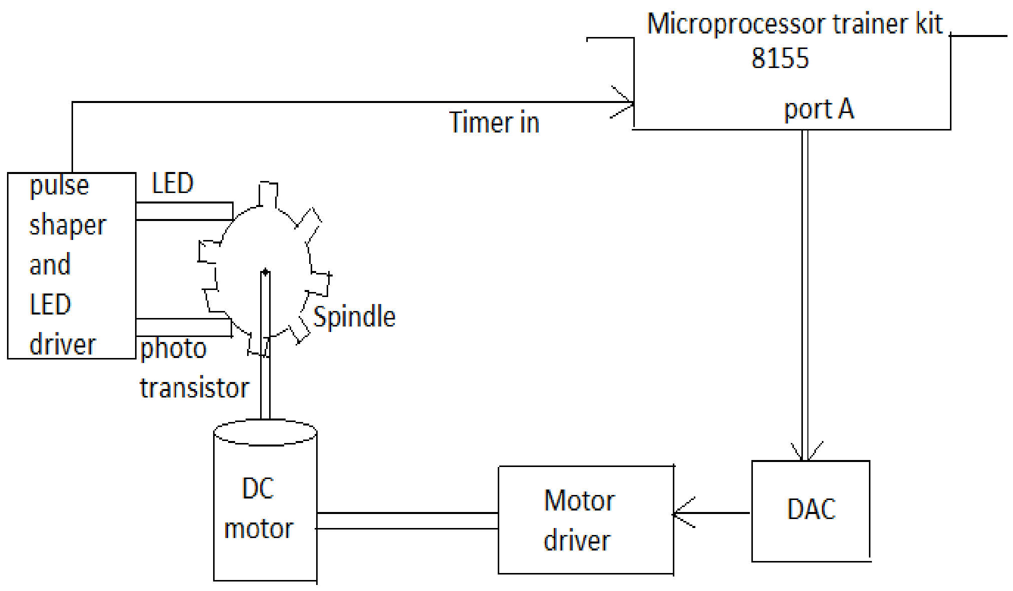


Fig.5.16

(ii) Speed Measurement:

i. To measure the speed of the motor a circular disk with several slots is attached to the spindle of the motor.
ii. An LED and photo transistor assembly (or an opto-coupler) is used to generate the pulse train with frequency related to the speed of the motor.
iii. For each rotation of the spindle the photo transistor is obstructed and exposed to the light emitted by the LED for n-times where n is the number of slots on the circular disk.
iv. By using suitable pulse shaper circuit, it is converted into a TTL comparable rectangular wave.
v. The rectangular wave is applied to a timer/ counter section to measure the frequency from which the speed can be determined.

(iii) Speed Control:

i. The speed is controlled by applying a variable DC voltage across the armature coil of the molar, thereby producing a variable armature current.
ii. The variable DC voltage is generated in the help of a DAC.
iii. A current amplifier is used to drive the motor.

(iv) I/O port and timer/counter:

For interfacing the extra hardware, only one 8-bit port and a timer/counter section are necessary. The port A and timer/counter of the 8155 available in the trainer kit is used.

(v) DAC and current amplifier:

DAC (Digital Analog Converter) is implemented by IC 1408. The output of DAC is applied to a motor driver buffer amplifier as shown Fig.5.17. The buffer amplifier is basically an emitter follows:

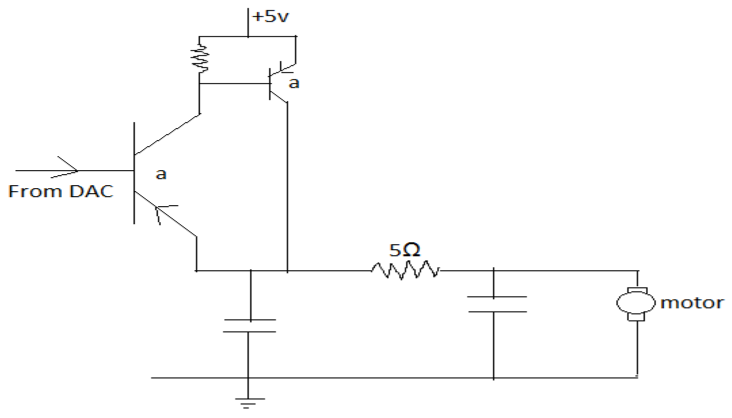


Fig.5.17

(vi) Pulse shaper and LED driver:

i. The LED permanently excited by a specified forward current to emit light.
ii. The signal generated by photo transistor is converted to a TTL compatible signal by using a Schmitt trigger circuit using transistor. The circuit diagram shown in Fig.5.18 .
iii. The output of the pulse shapes is fed to the clock input of the timer/counter section for the measurement of frequency.

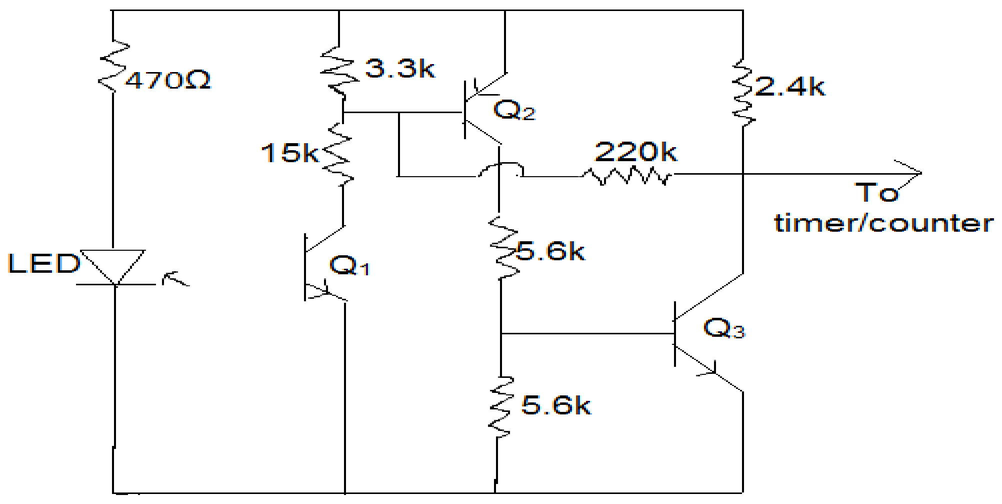


Fig.5.18

(vii) Display and keyboard:

i. For displaying the speed of the motor the data fields display of the drainer kit is used.
ii. Keyboard of the trainer kit is used for entiring the required speed.

(viii) Software design:

Software has to be designed to implement the following steps:

i. Read the desired speed from the keyboard.
ii. Choose an initial data to be applied to the DAC. So that the motor starts running.
iii. Apply data to the DAC through port A
iv. Generate delay for the motor to attain stable speed.
v. Measure the speed of the motor.
vi. Compare the speed with the required speed. If the speed matches then go to step 7. If the required speed is low and then increment data by 1, otherwise decrement data by 1 and go to step 7.
vii. Display the speed on the display

viii. Go to step 3.

Flow chart:

A flow chart of these operation in the proper sequence is shown in Fig.5.19.

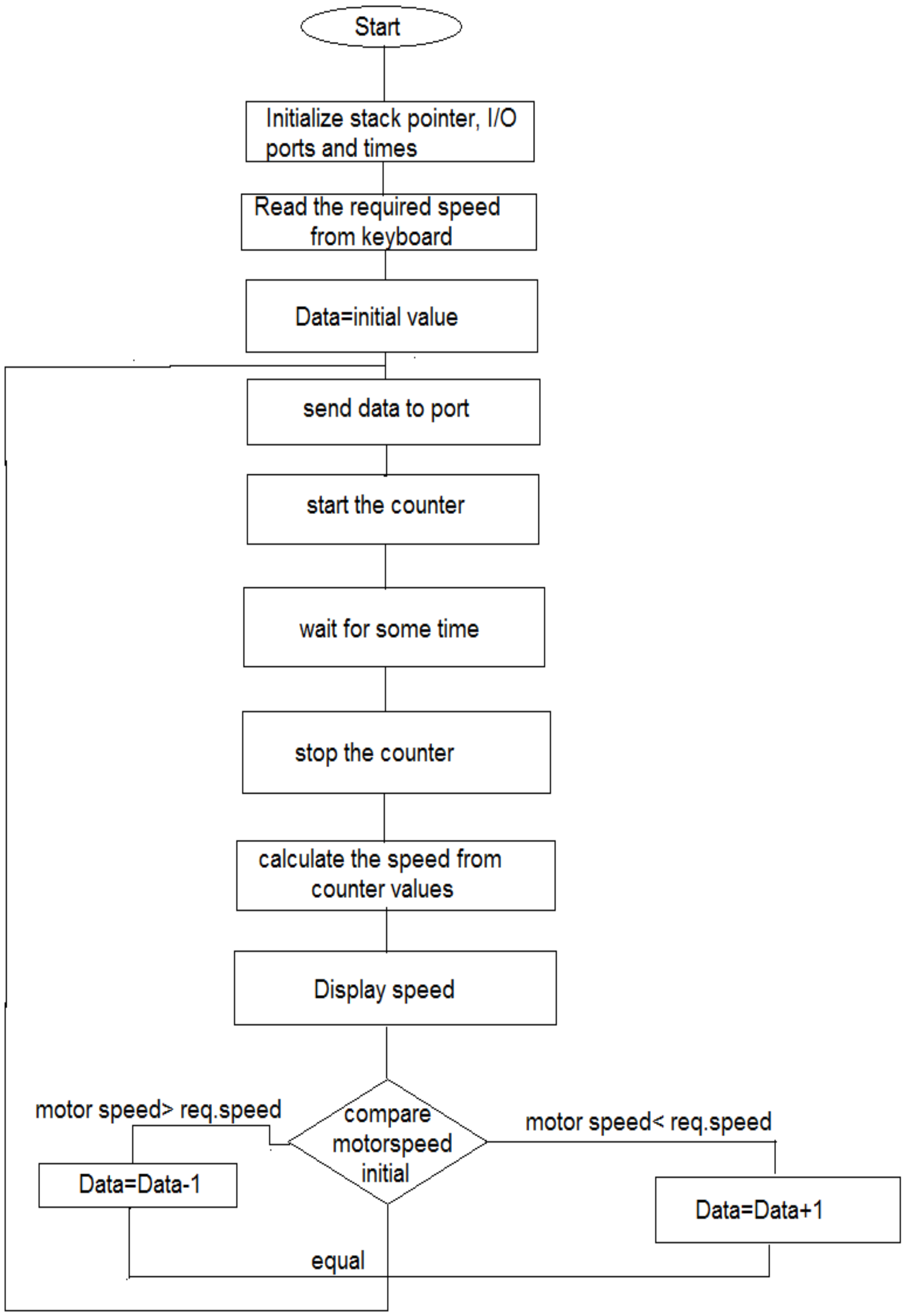


Fig.5.19

Program:

Assume the following monitor subroutines are available in μp trainer kit. The program of Dc motor speed control is shown in Fig.5.20

GETHEX:

Reads the keyboard and the stores hex , key entered in 'A' register .

DISDATA:

Display the content of 'A' register in the data field of the display.

DELAY:

Produces the required delay.

Label	Mnemonics	Comments
	LXI SP, 20FF	Initialize stack pointer
	MVI A,01	Control word for port A as output.
	OUT 28	send the cw to 8155
	CALL GETHEX	read keyboard character.
	MOV C,A	store it in 'c' register
	MVI D,CO	initial speed data
TOP	MOV A,D	
	OUT 29	send to port A, thereby to DAC
	CALL DELAY	wait for the speed up for motor
	MVI A,00	timer high port
	OUT 2D	send to timer high port
	MVI A,FF	timer low values. This value is decrement by the counter for each timer-in pulse

OUT 2C	send the cw to 8155
MVI A,C1	control word to start the times.
OUT 28	send the cw to 8155
CALL DELAY	wait for counting the pulses.
MVI A,41	control word to step the Times
OUT 28	send the cw to 8155
MOV B,A	save it in B register.

Fig.5.20

5.8 Traffic Light Control Using 8085:

One of the applications of microprocessor based systemis traffic light controller. By interfacing with microprocessor the signal lamps in a road junction are controlled using suitable hardware and software. Figure 5. 21 shows the arrangement of the lights in the road junction.

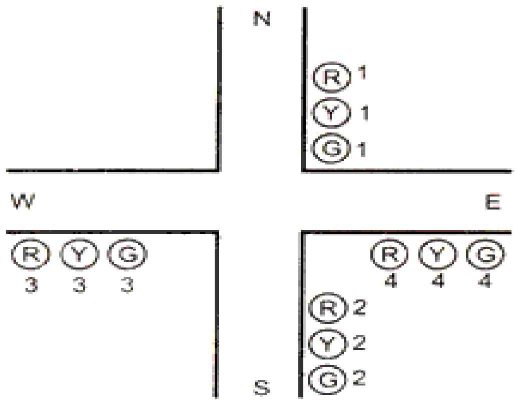


Fig.5.21 Hardware for traffic light control

(i) Delay program

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

- RET : Return to main program

Fig. 5.22 shows the interfacing diagram to control 12 electric bulbs. Port A is used to control lights on N-S road and Port B is used to control lights on W-E road. Actual pin connections are listed in Table 1 below.

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

Table 1

The electric bulbs are controlled by relays. The 8255 pins are used to control relay on-off action with the help of relay driver circuits. The driver circuit includes 12 transistors to drive 12 relays. Fig. 5.22 also shows the interfacing of 8255 to the system.

(ii) Interfacing diagram

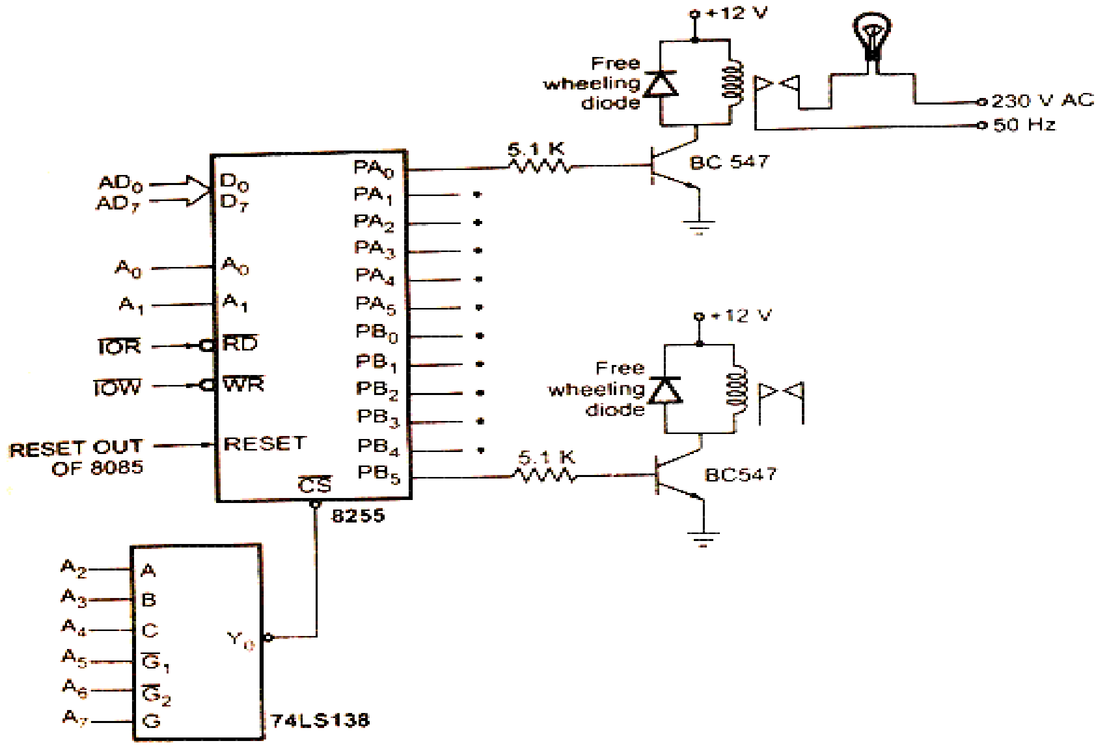
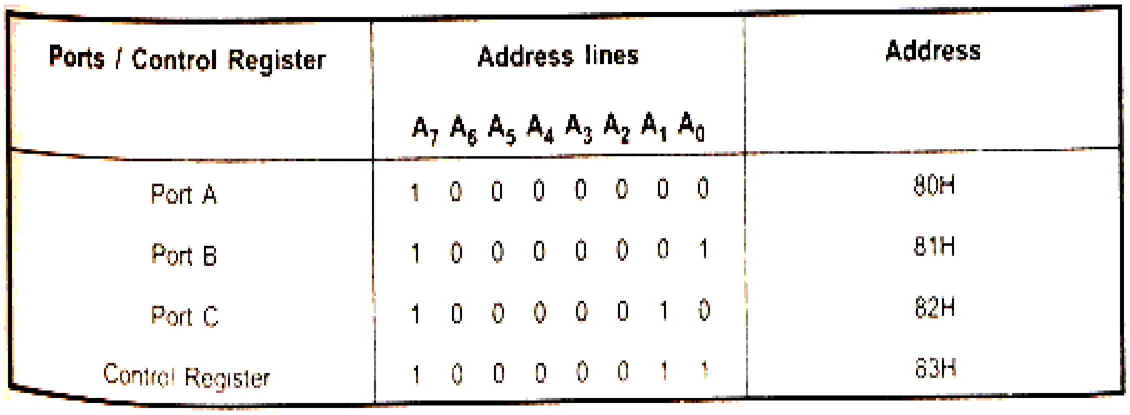


Fig. 5.22 Interfacing diagram

I/O MAP:

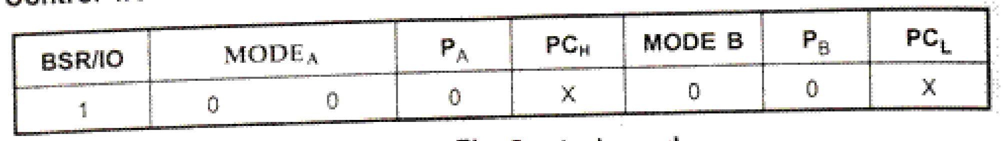


Ports / Control Register	Address lines A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0	Address
Port A	1 0 0 0 0 0 0 0	80H
Port B	1 0 0 0 0 0 0 1	81H
Port C	1 0 0 0 0 0 1 0	82H
Control Register	1 0 0 0 0 0 1 1	83H

Table 2

(iii) Software for traffic light control

Control word : For initialization of 8255.

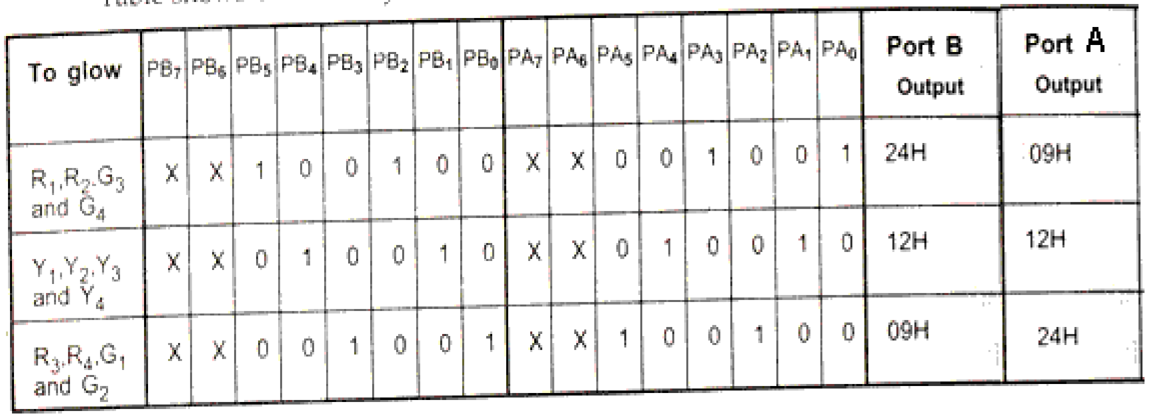


BSR/IO	$MODE_A$		P_A	PC_H	MODE B	P_B	PC_L
1	0	0	0	X	0	0	X

= 80H

Fig. Control word

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



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

Source Program 1:

- MVI A, 80H : Initialize 8255, port A and port B
- OUT 83H (CR) : in output mode
- START: MVI A, 09H
- OUT 80H (PA) : Send data on PA to glow R1 and R2
- MVI A, 24H
- OUT 81H (PB) : Send data on PB to glow G3 and G4

- MVI C, 28H : Load multiplier count (40_{10}) for delay
- CALL DELAY : Call delay subroutine
- MVI A, 12H
- OUT (81H) PA : Send data on Port A to glow Y1 and Y2
- OUT (81H) PB : Send data on port B to glow Y3 and Y4
- MVI C, 0AH : Load multiplier count (10_{10}) for delay
- CALL: DELAY : Call delay subroutine
- MVI A, 24H
- OUT (80H) PA : Send data on port A to glow G1 and G2
- MVI A, 09H
- OUT (81H) PB : Send data on port B to glow R3 and R4
- MVI C, 28H : Load multiplier count (40_{10}) for delay
- CALL DELAY : Call delay subroutine
- MVI A, 12H
- OUT PA : Send data on port A to glow Y1 and Y2
- OUT PB : Send data on port B to glow Y3 and Y4
- MVI C, 0AH : Load multiplier count (10_{10}) for delay
- CALL DELAY : Call delay subroutine
- JMP START

Delay Subroutine:

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

5.9 Interfacing of 7 segment LED display

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

(i) Hardware for eight digit seven segment display interface:

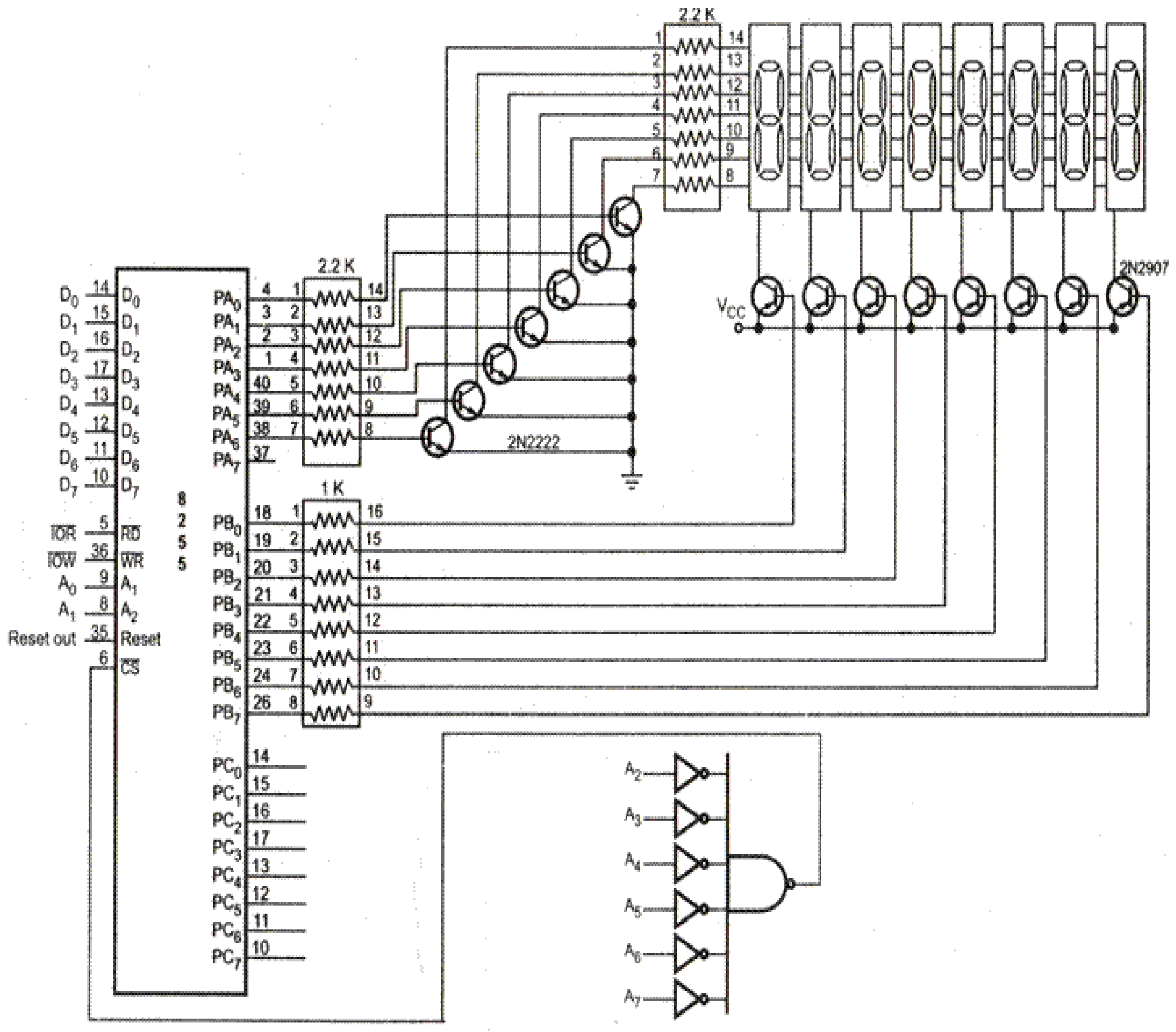


Fig.5.23 Interfacing of multiplexed eight 7-segment display using 8255

(ii) Software for eight digit seven segment display:

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

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

= 80 H

(iii) Source program:

- SOFTWARE TO INITIALIZE 8255:
- MVI A, 80H : Load control word in AL
- OUT CR : Load control word in CR

(iv) Subroutine to display message on multiplexed led display:

Set up registers for display:

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

Display message:

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

Note: This "display message subroutine" must be called continuously to display the 7-segment coded message stored in the memory from address 6000H.

Delay Subroutine:

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