CHAPTER – 4

MICROPROCESSOR SYSTEM

4.1 Memory Classification and Hierarchy

Memory is an essential component of the microcomputer system. It is used to store both instructions and data. It is used to store both instructions and data. Memory is made up of registers and the number of bits stored in a register is called memory word. Memory word is identified by an address. If microprocessor uses 16 bit address, then there will be maximum of $2^{16} = 65536$ memory addresses ranging from 0000H to FFFFH.

4.1.1 Classification of Memory

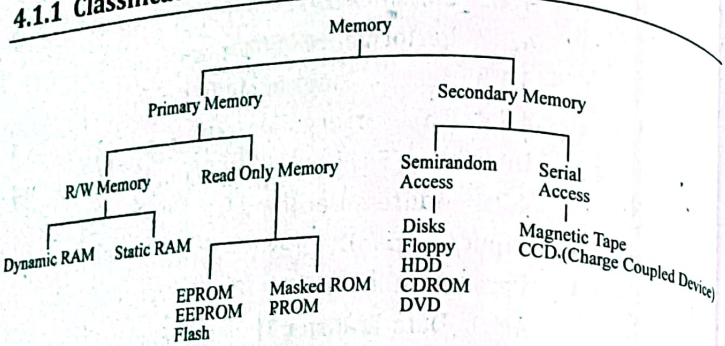

Figure 4.1: Classification of memory system

There are various types of memory which can be classified into two main groups: *primary memory* and *secondary memory*.

1. **Primary Memory**

Primary memory is the memory used by microprocessor to execute programs. The microprocessor can access only those items that are stored in this memory. Hence, all data and program must be within primary memory prior to its execution. Primary memory is much larger than processer memory that is included in the microprocessor chip.

Primary memory is divided into two groups:

i. **R/W Memory (RAM)**

Microprocessor can read from and write into this memory. This memory is used for information that are likely to be altered such as writing program or receiving data. This memory is volatile i.e., the content will be lost if the power is turned off, and is commonly known as RAM. RAM are basically of two types:

a. **Static RAM (SRAM):** This memory is made up of flip flops and it stores bit as voltage. A single flip flop stores binary data either 1 or 0. Each flip flop is called storage cell. Each cell requires six transistors. Therefore, the memory chip has low density but high speed. This memory is more expensive and consumes more power.

b. **Dynamic RAM (DRAM):** This memory is made up of MOS transistor gates and it stores the bit as charge. The advantage of DRAM are it has high density, low power consumption and cheaper than SRAM. But the bit information leaks therefore needs to be rewritten again every few milliseconds. It is called refreshing the memory and requires extra circuitry to do this. It is slower than SRAM.

ii. **Read Only Memory (ROM)**

ROM contains a permanent pattern of data that cannot be changed. It is non volatile that is no power source is required to maintain the bit values in memory. ROM are basically of 5 types:

a. **Masked ROM:** A bit pattern is permanently recorded by the manufactures during production.

b. **Programmable ROM:** In this ROM, a bit pattern may be written into only once and the writing process is performed electrically. That may be performed by a supplier or customer.

c. **Erasable PROM (EPROM):** This memory stores a bit in the form of charge by using EPROM programmer which applies high voltage to charge the gate .Information can be erased by exposing ultra violet radiation. It is reusable. The disadvantages are :

- It must be taken out of circuit to erase it

- The entire chip must be erased
- The erasing process takes 15 to 20 minutes.

d. **Electrically Erasable PROM (EEPROM):** It is functionally same as EPROM except that information can be altered by using electrical signal at the register level rather than erasing all the information. It is expensive compared to EPROM and flash and can be erased in 10 ms.

e. **Flash Memory:** It is variation of EPROM. The difference is that EPROM can be erased in register level but flash memory must be erased in register level but flash memory must be erased in its entirety or at block level.

2. **Secondary Memory**

The devices that provide backup storage are called *secondary memory*. It includes serial access type such as magnetic disks and random access type such as magnetic disks. It is nonvolatile memory.

4.1.2 Performance of Memory

1. **Access Time (t_a)**
   - **Read access time:** It is the average time required to read the unit of information from memory.
   - **Write access time:** It is the average time required to write the unit of information on memory.

   Access rate (r_a) = $1/t_a$

2. **Cycle Time (t_c)**

   It is the average time that lapses between two successive read operation.

   Cycle rate (r_c)= bandwidth = $1/t_c$

4.1.3 Access Modes of Memory

1. **Random access:** In *random access* mode, the t_a is independent of the location from which the data is accessed like MOS memory.

- Therefore, it is more ... low-cost storage devices to serve as a backup for storing the information that is not currently used by CPU.
- The memory unit that directly communicate with CPU is called the *main memory*.
- Devices that provide backup storage are called *auxiliary memory*.
- The memory hierarchy system consists of all storage devices employed in a computer system from the slow by high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory.
- The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices through an I/O processor.
- A special very-high-speed memory called *cache* is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.
- CPU logic is usually faster than main memory access time, with the result that processing speed is limited primarily by the speed of main memory.
- The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations.
- The memory hierarchy system consists of all storage devices employed in a computer system from slow but high capacity auxiliary memory to a relatively faster cache memory accessible to high speed processing logic. The figure below illustrates memory hierarchy.

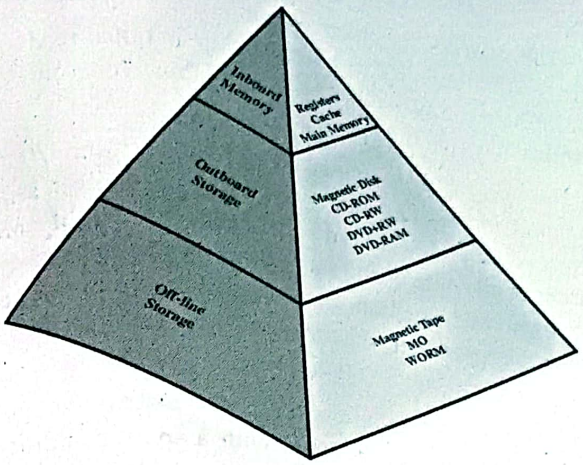

Figure 4.2: Memory Hierarchy

- As we go down in the hierarchy,
  - Cost per bit decreases
  - Capacity of memory increases
  - Access time increases
  - Frequency of access of memory by processor also decreases.

4.2 Interfacing I/O Devices and Memory with 8085

4.2.1 Address Decoding

The read/write memory is made of registers, and each register has a group of flip-flops that store bits of information; these flip-flops are called *memory cells*. The number of bits stored in a register is called a *memory word*. In a memory chip, all registers are arranged in a sequence and identified by binary numbers called *memory address*.

To communicate with memory, the processor should be able to:

- Select the chip
- Identify the register
- Read from or write into the register

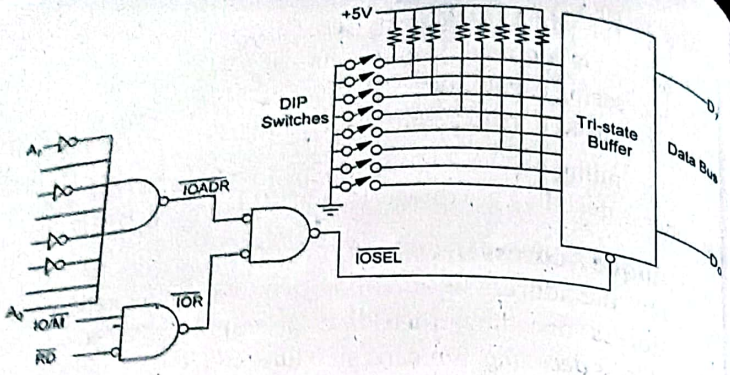

Figure 4.4: Input interfacing at 53H

b. Non-Unique Address Decoding

If all the address lines available on that mode are not used in address decoding, then that decoding is called *non-unique address decoding*. Though it is cheaper, there may be a chance of address conflict.

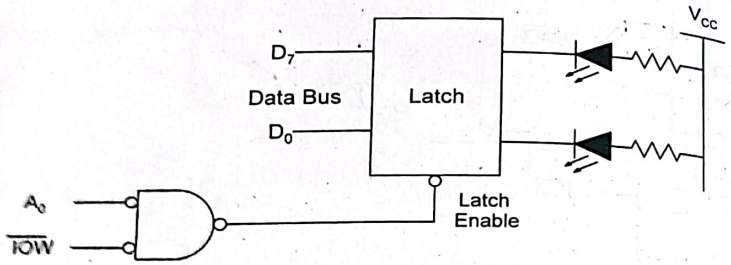

Figure 4.5: Non-unique address decoding

- If A_0 is low and $\overline{IOW}$ is low. Then latch gets enabled.
- Here A_1-A_7 is neglected that is any even address can enable the latch.

Examples:

1. **Design an interfacing circuit for following problem.**
   i. **74LS138: 3 to 8 Decoder**
   ii. **2732 (4K × 8): EPROM, address range should begin at 0000H and additional 4K memory space should be available for future expansion.**
   iii. **6116(2K × 8): CMOS R/W memory.**

To design a memory address decoding circuit, first calculate the number of address pins used by the memory device depending upon the memory capacity they can handle. Then determine the mapping addresses for memory devices (RAM, ROM). Then determine the different address pins which need to provide the input for n × 2^n decoder. Finally select the memory chips by using the appropriate decoder output line and required control signals (IO/ $\overline{M}$, $\overline{RD}$, $\overline{WR}$).

Let us draw a circuit diagram of a memory device interfacing where one 4K×8 EPROM chip, additional 4K x8 for future expansion and one 2K×8 CMOS R/W chip at address 0000H.

Step 1: Calculate the number of address pins

4K×8 chip requires 12 no. of address pins and 2K×8 chip requires 11 no. of address pins. This can be calculated from

n = log (memory capacity in bytes) / log (2)

Step 2: Memory mapping table

Memory Block	Address	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
EPROM	Start:0000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	End:0FFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
RAM	Start:2000H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	End: 27FFH	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1

The address select an address within memory chip or I/O chip and then read or write into it through the available data bus and thus avoid contention or data collision within the data bus.

Microprocessor is connected with memory and I/O devices via common address and data bus. Only one device can send data at a time and other devices can only receive that data. If more than one device sends data at the same time, the data gets garbled. In order to avoid this situation, ensuring that the proper device gets addressed at proper time, the technique called *address decoding* is used.

In address decoding method, all devices like memory blocks, I/O units, etc. are assigned with a specific address. The address of the device is determined from the way in which the address lines are used to derive a special device selection signal known as chip select ($\overline{CS}$). If the microprocessor has to write or to read from a device, the $\overline{CS}$ signal to that block should be enabled and the address decoding circuit must ensure that $\overline{CS}$ signal to other devices are not activated.

Depending upon the number of address lines used to generate chip select signal for the device, the address decoding is classified as:

I/O Mapped I/O

In this method, a device is identified with an 8-bit address and operated by I/O related functions IN and OUT for which $IO/\overline{M} = 1$. Since only 8-bit address is used, at most 256 bytes can be identified uniquely. Generally, lower order address bits (A_0-A_7) are used and higher order address bits (A_8-A_{15}) are considered don't care. Usually I/O mapped I/O is used to map devices like 8255A, 8251A, etc.

Memory Mapped I/O

In this method, a device is identified with 16-bit address and operated by memory related functions such as STA, LDA for which $IO/\overline{M} = 0$. Here, chip select signal of each device is derived from 16-bit address lines thus total addressing capability is 64K bytes. Usually memory mapped I/O is used to map memories like RAM, ROM, etc.

Depending on the address that are allocated to the device, the address decoding are categorized in the following two groups:

a. **Unique Address Decoding**

If all the address lines on that mapping mode are used for address decoding, then that decoding is called *unique address decoding*. It means all 8 lines in I/O mapped I/O and all 16 lines in memory mapped I/O are used to derive $\overline{CS}$ signal. It is expensive and complicated but fault proof in all cases.

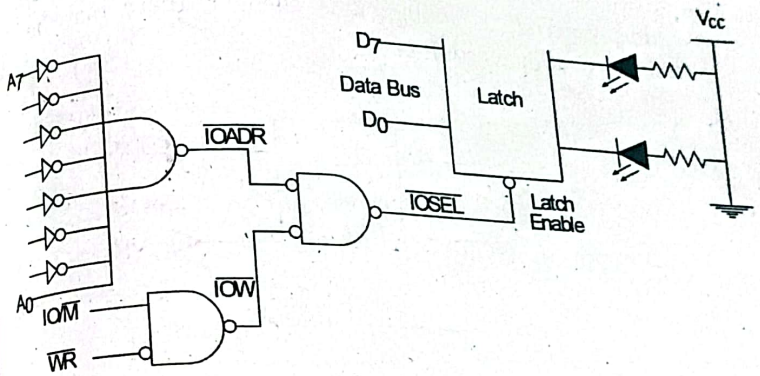

Figure 4.3: Unique address decoding

- If A_0 is high and A_1- A_7 are low and if $\overline{IOW}$ becomes low, the latch gets enabled.
- The data to the LED can be transferred in only one case and hence the device has unique address of 01H.

Eight I/P switch interfacing at 53H (01010011):

Here, after placing EPROM, we need to reserve 4K ×8 memory for future expansion; hence starting address of RAM comes to 2000H instead of 1000H.

Step 3: Decide decoder pins

Here, bits A_{11} and A_{13} address lines are different for 2 chips referring to end address lines. We need to use a 3×8 decoder, so we need to add one address line and here we can use A_{12}. We can use address lines A_{14} and A_{15} to generate chip enable signals for 3×8 decoder.

Step 4: Draw a decoding circuit

The output from decoder Y_1 (001) selects the EPROM and Y_4 (100) selects RAM. The address decoding circuit is drawn below.

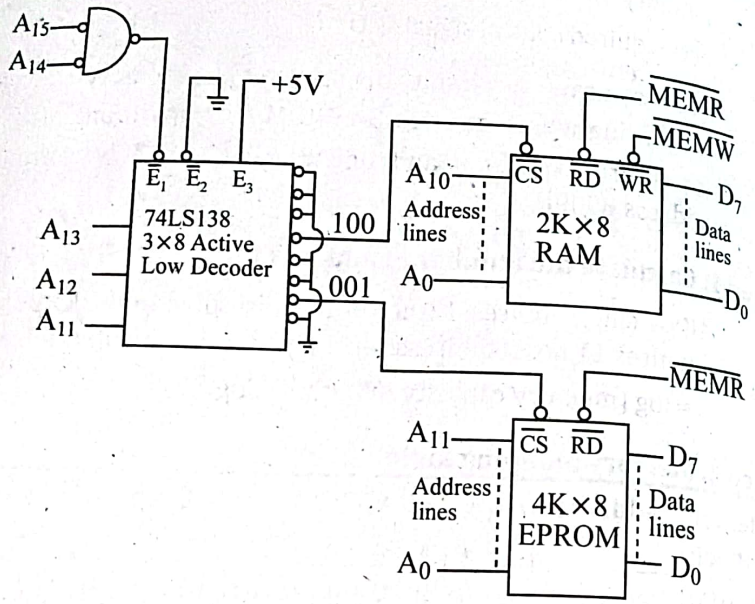

Fig.: Address decoding circuit

Design an interfacing circuit to interface one 4KB EPROM and two 2KB R/W memory for 8085 microprocessor.

To design a memory address decoding circuit, first calculate the number of address pins used by the memory device depending upon the memory capacity they can handle. Then determine the mapping addresses for memory devices (RAM, ROM). Then, determine the different address pins which need to provide the input for n× 2^n decoder. Finally, select the memory chips by using the appropriate decoder output line and required control signals (IO/$\overline{M}$, $\overline{RD}$, $\overline{WR}$).

Drawing a circuit diagram of a memory device interfacing with one 4KB EPROM chip, and two 2KB RAM chips at address 8000H involves the following steps.

Step 1: Calculate the number of address pins

4K×8 chip requires 12 no. of address pins and 2K×8 chip requires 11 no. of address pins. This can be calculated from

n = log (memory capacity in bytes) / log (2)

Step 2: Memory mapping table

Memory Block	Address	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
EPROM	Start:8000H	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	End:8FFFH	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
RAM1	Start:9000H	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	End: 97FFH	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1
RAM2	Start:9800H	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
	End: 9FFFH	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

Step 3: Decide decoder pins

Here, bits A_{11}, A_{12} and A_{13} address lines are different for 3 chips referring to end address lines. So, we need to use a

3×8 decoder. We ... A_{14} and A_{15} to generate chip enable signals for 3×8 decoder.

Step 4: Draw a decoding circuit

The output from decoder Y_1 (001) selects the EPROM, Y_2 (010) selects RAM1 and Y_3 (011) selects RAM2 chips. The address decoding circuit is drawn below.

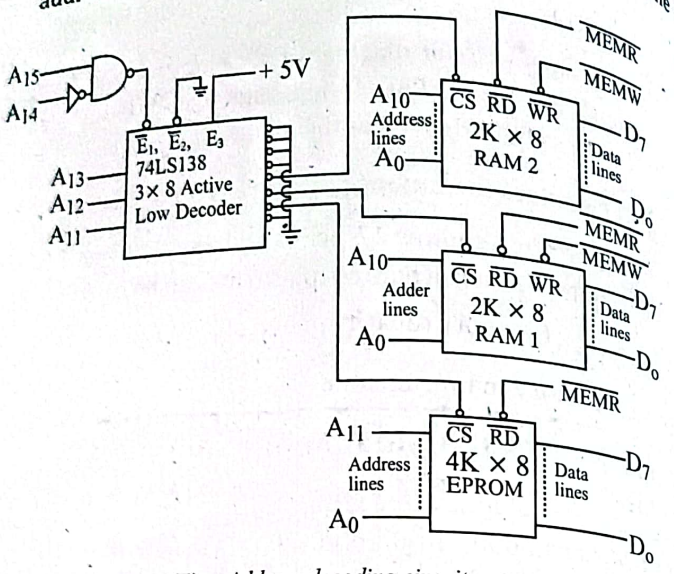

Fig.: Address decoding circuit

3. With a neat diagram, explain the interfacing circuit using a 3:8 decoder (74LS138) needed to connect the following memory units to the 8085 microprocessor consecutively starting from memory location A000H.

i. 2K×8 ROM chip

ii. 2K×8 RAM chip

iii. 2K×8 EPROM chip

To design a memory address decoding circuit, first calculate the number of address pins used by the memory device depending upon the memory capacity they can handle. Then determine the mapping addresses for memory devices (RAM, ROM). Then, determine the different address pins which need to provide the input for n×2ⁿ decoder. Finally, select the

memory ... the appropriate decoder output line and required control signals (IO/$\overline{M}$, $\overline{RD}$, $\overline{WR}$).

Drawing a circuit diagram of a memory device interfacing with one 2K×8 ROM chip, 2K×8 RAM chip and one 4K ×8 EPROM chip at address A000H involves the following steps.

Step 1: Calculate the number of address pins

2K×8 chip requires 11 no. of address pins and 4K×8 chip requires 12 no. of address pins. This can be calculated from

n = log (memory capacity in bytes) / log (2)

Step 2: Memory mapping table

Memory Block	Address	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
ROM	Start:A000H	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	End:A7FFH	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1
RAM	Start:A800H	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
	End: AFFFH	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1
EPROM	Start: B000H	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	End: BFFFH	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Step 3: Decide decoder pins

Here, bits A_{11} and A_{12} address lines are different for 3 chips referring to start or end address lines. We need to use a 3×8 decoder, so we need to add one address line and here we can use A_{13}. We can use address lines A_{14} and A_{15} to generate chip enable signals for 3×8 decoder.

Step 4: Draw a decoding circuit

The output from decoder Y_4 (100) selects the ROM, Y_5 (101) selects RAM and Y_6 (110) selects the EPROM. The address decoding circuit is drawn below.

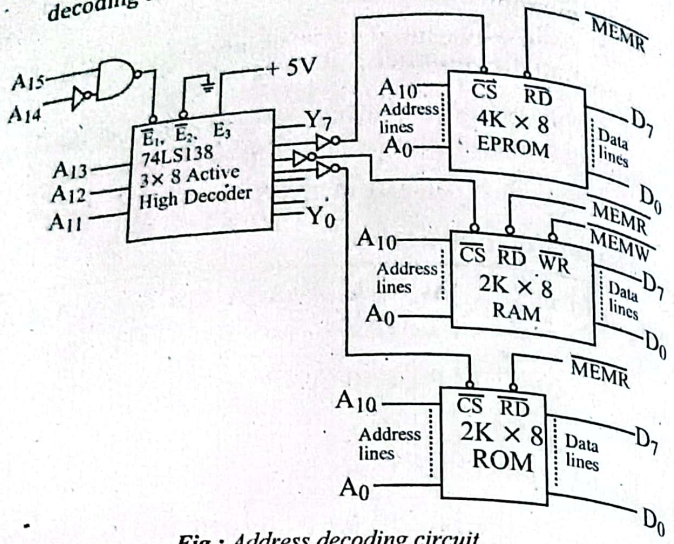

Fig.: Address decoding circuit

4.3 Input/Output Devices

Input/output devices are the means through which the microcomputer unit communicates with the outside world. The link between the I/O devices and the microprocessor is maintained by a circuitry known as I/O module. This circuitry includes the specific interfaces needed for I/O devices as well as control functions that implement the I/O transfers within the computer. I/O devices usually are appeared as passive devices which take action only when instructed to do. The CPU monitors the status of the I/O devices and selects them according to availability and need.

Consider the keyboard as input device and the steps when the key is pressed are:

- Microprocessor detects the key change in status of keyboard i.e., the key is pressed.

- It receives the encoded information corresponding to pressed key.
- It checks the validity of required signal.

Consider the printer as output device.

- Here, microprocessor checks ideal condition of printer, if ideal then sends the data to be printed and required command for that.

For interfacing of typical microprocessor to I/O devices such as keyboard, CRT, printer etc., all need I/O interface circuits which are of mainly two types: *parallel interface* and *serial interface*.

4.4 Parallel Interface

A *parallel interfacing* refers to a multiline channel, which is capable of transmitting several bits of data simultaneously. This means, parallel interfacing transfers n-bit data at the same time, using n number of wires. In parallel interfacing with microprocessor-based system, microprocessor and input/output devices are interconnected with multiline channel allowing a system to achieve higher data rate and hence maximizes the bandwidth. Before USB ports became common, most personal computers (PCs) had at least one parallel interface for connecting a printer using a parallel port.

4.4.1 Data Transfer Modes of Parallel Interfacing

Parallel transmission of data is used for short distance where the speed of information transfer is critical. This form of data communication is found in newer type of computer peripheral equipment with transfer speed of to one million characters per second. The equipment includes printers, disk drives, and various other forms of peripheral components.

The information exchanged between a microprocessor and an I/O interface circuit consists of input or output data and control information. The status information enables the microprocessor to monitor the device and when it is ready then send or receive data. Control information is the command by

microprocessor action. If the device operates at different speeds, then microprocessor can be used to select a particular speed of operation of the device. The techniques used to transfer data between different speed devices and computer is called synchronizing. There are various ways of synchronization techniques which are involved in parallel data transfer such as *simple input and output, simple strobe I/O, single handshaking*, and *double handshaking*.

1. **Simple I/O**

To get digital data from a simple switch into a microprocessor; switch is connected on input port line from which port can be read. The data is always present and ready so that it can be read at any time. Similarly, to output data to a simple display device like LED, the input of LED buffer is connected on an output port pin. And output the logic level required turning on the light. The LED is always there and ready so that data can be sent at any time.

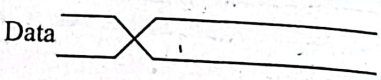

Figure 4.6: Simple I/O

This timing waveform illustrates the simple I/O where cross lines represent the time at which a new data byte becomes valid on the output lines of the port. Absences of other waveforms indicate that this output operation is not directly dependent on any other signals.

Simple Strobe I/O

In many applications, valid data is present on an external device only at a certain time and must be read in at that time. Here a strobe pulse is supplied to indicate the time at which data is being transmitted. For an example, we can discuss the ASCII encoded keyboard. When a key is pressed, circuitry on keyboard sends out ASCII code for pressed key on eight parallel data lines and then sends out a strobe signal on another line to indicate that valid data is present on eight data lines

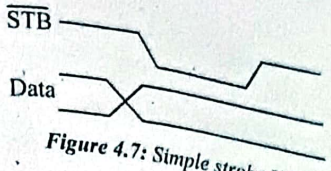

Figure 4.7: Simple strobe I/O

The sending device outputs parallel data on the data lines, and then outputs $\overline{STB}$ signal to represent the valid data is present. In this technique, microprocessors need to wait until the device is ready for the operation and also known as *simple wait I/O*.

Single Handshaking

2. *Handshaking* is the method of synchronizing the actions of slow peripheral devices with that of high speed microprocessor. It can have two transfer schemes:

Input handshake (peripheral to microprocessor):

The peripheral outputs some data and sends some strobe signal to microprocessor. Microprocessor detects asserted strobe signal ($\overline{STB}$) and reads the byte of the data. Processor then sends acknowledgement signal (ACK) to peripheral to indicate that the data has been read and can send next byte of data.

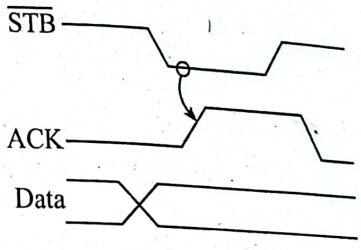

Figure 4.8: Single handshaking

- The peripheral outputs some data and send $\overline{STB}$ signal to microprocessor to tell "Here is the data for you".
- Microprocessor detects asserted $\overline{STB}$ signal, reads the data and sends an acknowledge signal (ACK) to indicate 'data has been read and peripheral can send next data, "I got that one, send me another".

- Microprocessor ... data when peripheral is ready.

Output handshake (peripheral from microprocessor):

Microprocessor outputs data to peripheral and asserts a strobe ($\overline{STB}$) signal. If peripheral is ready, it answers back with acknowledgement (ACK) signal to microprocessor.

4. **Double Handshaking**

For data transfers where even more coordination is required between the sending system and the receiving system, a *double handshake* is used. It can have two transfer schemes.

Input handshake (peripheral to microprocessor):

Peripheral asserts strobe ($\overline{STB}$) line low to ask receiving device whether it is ready or not for data reception. Receiving system raises its acknowledgement (ACK) line high to indicate it is ready. Peripheral device then sends the byte of data and raises its strobe ($\overline{STB}$) line high. When microprocessor reads data, it drops its acknowledgement (ACK) line low and requests sending system to send next byte of data.

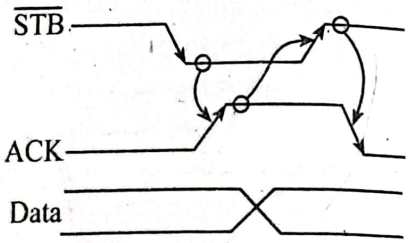

Figure 4.9: Double handshaking

- The peripheral asserts its $\overline{STB}$ line low to ask microprocessor "Are you ready?"
- The microprocessor raises its ACK line high to say "I am ready".
- Peripheral then sends data and raises its $\overline{STB}$ line low to say "Here is some valid data for you."

- Microprocessor then reads the data and drops its ACK line to say, "I have the data, thank you, and I await your request to send the next byte of data."

Output handshake (peripheral from microprocessor):

Microprocessor sends a strobe ($\overline{STB}$) signal and data and peripheral sends acknowledgement (ACK) signal.

4.4.2 Introduction to Programmable Peripheral Interface (PPI) - 8255A

The Intel 8255A is a general purpose programmable I/O device designed for use with Intel microprocessors. It has 24 I/O pins that can be grouped primarily in two 8-bit parallel ports: A and B, with the remaining bits as port C. The 8 bits of port C can be used as individual bits or be grouped in two 4-bits ports: C upper (C_U) and C lower (C_L). The functions of these ports are defined by writing a control word in the control register.

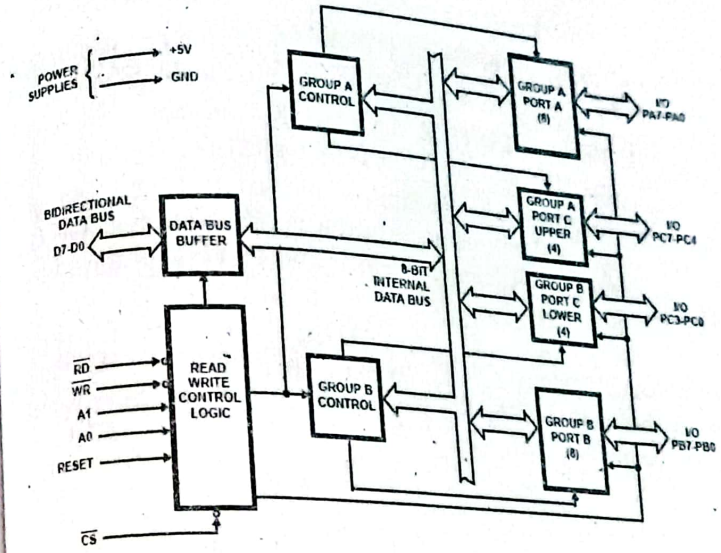

Figure 4.10: Internal block diagram of 8255A

The 8255A has the following main blocks:

a. **Data bus buffer**

The 3-state bidirectional 8-bit buffer is used to interface the 8255A to the system data bus. Data is transmitted o received by the buffer upon execution of input or outpu

instructions ... information are also transferred through the data bus and status buffer.

b. **Read/write control logic**

The function of this block is to manage all of the internal and external transfers of both data and control or status words. It accepts inputs from the CPU address and control buses and in turn, issues commands to both of the control groups.

- **Chip select ($\overline{CS}$):** A "low" on this pin enables the communications between the 8255A and the CPU.
- **Read ($\overline{RD}$):** A "low" on this input enables the 8255A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to read from the 8255A.
- **Write ($\overline{WR}$):** A "low" on this input pin enables the CPU to write data or control words into the 8255A.
- **Reset (RESET):** A "high" to this pin clears the control register and sets all ports (A, B and C) in the input mode.
- **A_0 and A_1:** These input signals control the selection of one of the three ports or the control word register. They are connected to the least significant bits of the address bus.

The $\overline{CS}$ signal is the master chip select, and A_0 and A_1 specify one of the I/O ports or the control register as given below.

Table 4.1: Port selection of 8255A

$\overline{CS}$	A_1	A_0	Selection
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control register
1	x	x	8255A is not selected

c. **Group A and group B controls**

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

- Control group A – Port A and port C upper (C_7–C_4)
- Control group B – Port B and port C lower (C_3–C_0)

The 8255A functions in two modes:

- **Bit set/reset mode:** The BSR mode is used to set or reset the bits in port C.
- **I/O mode:** The I/O mode is further divided into three modes: mode 0, mode 1, and mode 2. In mode 0, all ports function as simple I/O ports. Mode 1 is a handshake mode whereby ports A and/or B use bits from port C as handshake signals. In the handshake mode, two types of I/O data transfer can be implemented: status check and interrupt. In mode 2, port A can be set up for bidirectional data transfer using handshake signals from port C and port B can be set up either in mode 0 or mode 1.

The 8255A is a widely used, programmable, parallel I/O device. The features of 8255A are:

- It can be programmed to transfer the data under various conditions – from simple I/O to interrupt I/O.
- It is compatible with all Intel and most other microprocessors.
- It is completely TTL compatible.
- It has three 8-bit ports namely port A, port B, and port C which are arranged in two groups of 12 pins. Each port has a unique address and data can be read from or written to a port. In addition to the address assigned to the 3 ports, another address is assigned to the control register into which control words are written for programming the 8255 to operate in various modes.

- Its bit set/reset mode allows setting and resetting of individual bits of port C.
- The 8255A can operate in three I/O modes namely mode 0, mode 1, and mode 2.

8255 Control Word

When A_0 and A_1 pins have value 1, the mapped address addresses the control register which is the 8-bit register to write the specific content according to the port conditions although it cannot be read. The content of this register is called *control word* which specifies an I/O function for each port.

The MSB (D_7) of the control word tells which control word we are sending it, that is, it specifies either the I/O function or the bit set reset function. If bit D_7 = 1, bits D_6–D_0 determine I/O functions in various modes as shown in figure. If bit D_7 = 0, port C operates in the bit set/reset (BSR) mode. The BSR control word does not affect the functions of ports A and B.

I/O control word:

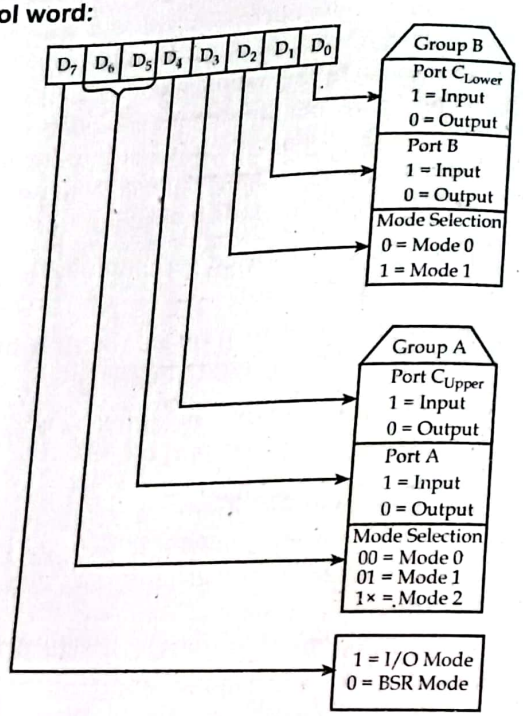

Figure 4.11: 8255A control word format for I/O mode

BSR control word:

This control word, when written in control register, sets or resets one bit at a time, as specified in Figure 4.34.

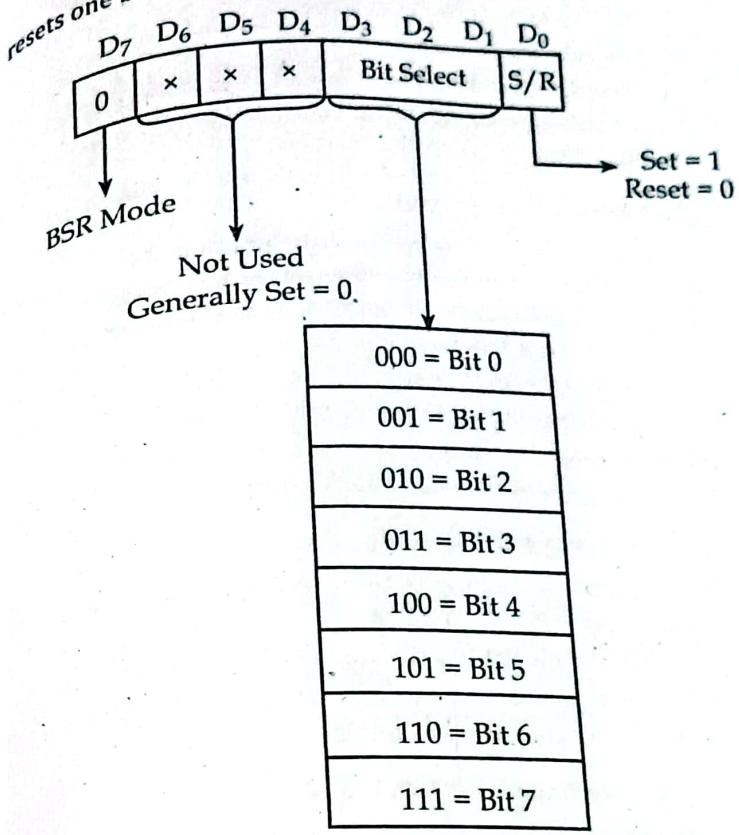

Figure 4.12: 8255A control word format for BSR mode

Operating Modes of the Intel 8255A

1. Mode 0 (Basic Input/Output)

This functional configuration provides simple input and output operation for each of the three ports. No 'handshaking" is required; data is simply written to or read from a specified port.

Mode 0 basic functional definitions:

- Two 8-bit ports (port A and port B) and two 4-bit ports (port C lower and port C upper) can operate in mode 0.
- Any port can be input or output
- Outputs are latched
- Inputs are not latched
- 16 different input/output configurations are possible in this mode.

2. **BSR Mode (Bit Set/Reset)**

BSR mode is concerned only with eight bits of port C, which can be set or reset by writing an appropriate control word in the control register. A control word with bit $D_7 = 0$ is recognized as a *BSR control word* and it does not alter any previously transmitted control word with bit $D_7 = 1$; thus the I/O operations of ports A and B are not affected by a BSR control word. In the BSR mode, individual bits of port C can be used for applications such as on/off switch.

3. **Mode 1 (Strobe Input/Output)**

The functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or handshaking signals. In mode 1, port A and port B use the lines of port C to generate or accept these handshaking signals.

Mode 1 basic functional definitions:

- Two ports, port A and port B can operate in mode 1.
- The 8-bit data ports (port A and port B) can be either input or output. Both inputs and outputs are latched.
- The separate 3 bits from port C are used to provide control and status for 8-bit data ports.

Mode 2 (Strobe Bidirectional Bus I/O)

The functional configuration provides a means for communicating with a peripheral device or a structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus I/O). "Handshaking signals" are provided

to maintain proper bus flow discipline in a similar manner to mode 1. Interrupt generation and enable/disable functions are also available.

Mode 2 basic functional definitions:

- Only port A can operate in mode 2.
- The 8-bit data port (port A) can be either input or output. Both inputs and outputs are latched.
- The 5 bits from port C are used to provide control and status for 8-bit data port.

4.5 Serial Interface/Serial Data Transmission

Within a microcomputer, data is transferred in parallel, because that is the fastest way to do it. For transferring data over long distances, however, parallel data transmission requires too many wires. Therefore, data to be sent long distances is usually converted from parallel form to serial form so that it can be sent on a single wire or pair of wires. Serial data received from a distant source is converted to parallel form so that it can easily be transferred on the microcomputer buses.

4.5.1 Serial Data Transmission

In a *serial data transmission*, the data are sent one bit at a time over the transmission channel. However, since most processors process data in parallel, the transmitter needs to transform incoming parallel data into serial data and the receiver needs to do the opposite.

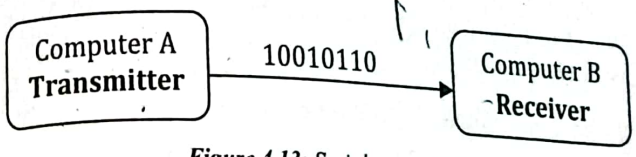

Figure 4.13: Serial transmission

In case of serial transmission, data is sent in a serial form i.e., bit by bit on a single line. Also, the cost of communication hardware is considerably reduced since only a single wire or channel is required for the serial bit transmission. Serial data transmission is slow as compared to parallel transmission.

Advantages of serial data transfer over parallel

- Longer data transmission in serial mode
  Serial: 1 → −3V to −25V
  0 → +3V to +25V
  Parallel: 1 → +5V
  0 → 0V
  Voltage loss is not much a problem in serial communication.
- Serial transmission requires less number of wires than parallel and so cheaper to transmit data.
- Crosstalk is less of an issue because there are fewer conductors compared to that of parallel cables.
- Many ICs and peripherals have serial interfaces.
- Clock skew between different cables is not an issue.
- Serials can be clocked at higher data rate.
- Serial cable can be longer than parallel.
- Cheaper to implement.

Serial data systems may be *simplex*, *half-duplex*, and *full-duplex*. A simplex data line can transmit data only in one direction (a commercial radio station is an example). Half-duplex transmission means that the communication can take place in either direction between two systems, but can only occur in one direction at a time (an example is a two-way radio system, where one user always listens while the other talks). The term "full-duplex" means that each system can send and receive data at the same time (a normal phone conversation is an example). Serial data can be sent synchronously or asynchronously.

1. Serial Synchronous Data Transmission

In *serial synchronous data transmission*, data is transmitted or received based on a clock signal. At a specific rate of data transmission, the transmitting device sends a data bit at each clock pulse. In order to interpret the data correctly, the receiving device must know the start and end of each data unit. The transmitter must know the number of data units to be transferred and the receiver must be synchronized

with the data boundaries. Therefore, there must be synchronization between the transmitter and receiver. Usually one or more sync characters are used to indicate the start of each synchronous data stream or frame of data. Transmitter sends a large block of data characters one after the other with no time between characters. Transmitting device sends data continuously to the receiving device. If the data is not ready to be transmitted, the line is held in marking condition. To indicate the start of transmission, the transmitter sends out one or more sync characters or a unique bit pattern called a flag, depending on the system being used. The receiving device waits for data, when it finds the sync characters or the flag then starts interpreting the data which shifts the data following the sync characters and converts them to parallel form so they can be read in by a computer.

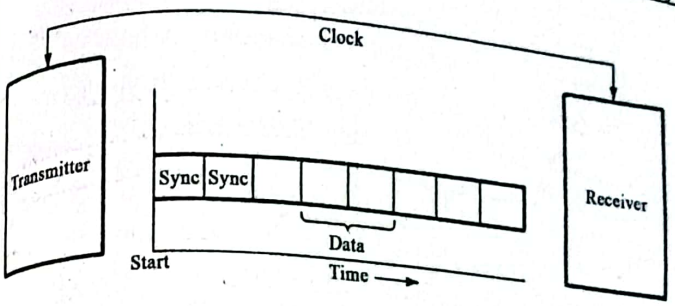

Figure 4.14: Synchronous serial transmission format

Synchronous transmission has the advantage that the timing information is accurately aligned to the received data, allowing operation at much higher data rates. It also has the advantage that the receiver tracks any clock drift which may arise (for instance due to temperature variation). The penalty is however a more complex interfaces design, and potentially a more difficult interface to configure (since there are many more interface options).

Data transmission takes place without any gap between two adjacent characters. However, data is sent block by block. A block is a continuous stream of characters or data bit

pattern between any two blocks of data and hence the data transmission is synchronized. Synchronous communication is used generally when two computers are communicating to each other at a high speed or a buffered terminal is communicating to the computer.

Advantages and disadvantages of synchronous communication:

Main advantage of synchronous data communication is the high speed. The synchronous communications require high-speed peripherals/ devices and a good-quality, high bandwidth communication channel.

The disadvantage includes the possible inaccuracy. Because when a receiver goes out of synchronization, loosing tracks of where individual characters begin and end, correction of errors takes additional time.

2. **Serial Asynchronous Data Transmission**

The receiving device does not need to be synchronized with the transmitting device. The transmitting device can send one or more data units when it is ready to send data. Each data unit must be formatted i.e., must contain start and stop bits for indicating beginning and the end of data unit. It also includes one parity bit to identify odd or even parity of data. To send ASCII character, the framing of data should contain:

- 1 start bit: Beginning of data
- 7 or 8-bit character: Actual data transferred
- 1 parity bit: Parity
- 1 or 2 stop bits: End of data

When no data is being sent, the signal line is in a constant high or marking state. The beginning of the data character is indicated by the line going low for 1-bit time and this bit is called a start bit. The data bits are then sent out on the line one after the other where the least significant bit is sent out first. Parity bit should contain to check for errors in received data. After the data bit and a parity bit, the signal

line is returned high for at least 1-bit time to identify the end of the character, this always high bit is referred to as a stop bit. Some older systems use 2 stop bits.

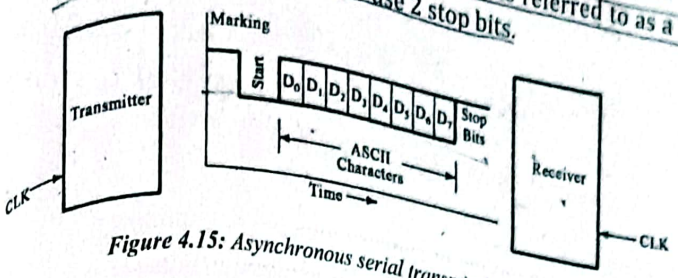

Figure 4.15: Asynchronous serial transmission format

In *asynchronous transmission*, each character is transmitted separately, that is one character at a time. The character (8-bits) is preceded by a start bit (1-bit), which tells the receiving end where the character coding begins, and is followed by a stop bit (1 or 2-bits), which tells the receiver where the character coding ends. There will be intervals of ideal time on the channel shown as gaps. Thus there can be gaps between two adjacent characters in the asynchronous communication scheme. In this scheme, the bits within the character frame (including start, parity and stop bits) are sent at the baud rate.

The *start* bit and *stop* bit including gaps allow the receiving and sending computers to synchronize the data transmission. Asynchronous communication is used when slow speed peripherals communicate with the computer. The main disadvantage of asynchronous communication is slow speed transmission. Asynchronous communication however, does not require the complex and costly hardware equipments as is required for synchronous transmission.

Synchronous Versus Asynchronous Serial Data Transmission

S.N.	Parameter	Asynchronous	Synchronous
1.	Fundamental	Transmission is not based on clock signal	Transmission based on clock signal
2.	Data Format	One character at a time	Group of characters i.e., a block of characters

S.N.	Pa		ous
3.	Speed		High (> 20 Kbps)
4.	Framing Information	Start and stop bits are sent with each character.	Sync characters are sent with each character.
5.	Implementation	Hardware/software	Hardware

Bit and Baud Rate

The difference between *bit rate* and *baud rate* is complicated and intertwining. Both are dependent and inter-related.

- **Bit rate** is how many data bits are transmitted per second.
- **Baud rate** is the number of times per second a signal in a communications channel changes.

Bit rates measure the number of data bits (that is 0's and 1's) transmitted in one second in a communication channel. A figure of 2400 bits per second means 2400 zeros or ones can be transmitted in one second, hence the abbreviation "bps." Individual characters (for example, letters or numbers) that are also referred to as bytes are composed of several bits.

A baud rate is the number of times a signal in a communications channel changes state or varies. For example, a 2400 baud rate means that the channel can change states up to 2400 times per second. The term "change state" means that it can change from 0 to 1 or from 1 to 0 up to X (in this case, 2400) times per second. It also refers to the actual state of the connection, such as voltage, frequency, or phase level).

The main difference between the two is that one change of state can transmit one bit, or slightly more or less than one bit, that depends on the modulation technique used. So, the bit rate (bps) and baud rate (baud per second) have this connection:

If signal is changing every $\frac{10}{3}$ ns, then

$$\text{Baud rate} = \frac{1}{\frac{10}{3}\text{ ns}} = \frac{3}{10}\times 10^9 = 3\times 10^8 = 300 \text{ Mbd}$$

Note: If 1 frame of data is coded with 1 bit, then baud rate and bit rate are same. When frame of data are coded with two or more bits, then baud rate and bit rate are not same.

4.5.2 Standards in Serial I/O

The *serial I/O technique* is commonly used to interface different peripheral terminals such as printers, modems with microcomputers which are designed and manufactured by various manufacturers. Therefore, a common understanding must exist among various manufacturing and user groups that can ensure compatibility among different equipment. The standard is defined as the understanding which is accepted in industry and by users. A standard is normally defined by professional organizations such as IEEE (Institute of Electrical and Electronics Engineers), EIA (Electronic Industries Association) as a de jure standard. However, a widespread practice can become a de facto standard.

In serial I/O, data can be transmitted as either current or voltage. When data are transmitted with current signal such for teletype equipment, 20 mA (or 60 mA) current loops are used. When a teletype is marking or at logic 1, current flows; when it is at logic 0 (space), the current flow is interrupted. The advantage of the current loop method is that signals are relatively noise-free and are suitable for transmission over a distance.

When data are transmitted with voltage signal, there are various standards which are explained in this section.

1. **RS-232C**

Serial transmission of data is used as an efficient means for transmitting digital information across long distances, the existing communication lines usually the telephone lines can be used to transfer information which saves a lot of hardware. RS-232C is an interface developed to standardize the interface between data terminal equipment (DTE) and data communication equipment (DCE) employing serial binary data exchange. Modem and other devices used to send serial data are called data communication equipment (DCE). The computers or terminals that are sending or

receiving the ... equipment (DTE).

Figure 4.16: RS-232C DTE and DCE connector plugs

RS-232C is the interface standard developed by electronic industries Association (EIA) in response to the need for the signal and handshake standards between the DTE and DCE. RS-232C has following standardize features:

- It uses 25 pins (DB – 25P) or 9 Pins (DE – 9P) standard where 9 pins standard does not use all signals i.e. data, control, timing and ground.
- It describes the voltage levels, impendence levels, rise and fall times, maximum bit rate and maximum capacitance for all signal lines.
- It specifies that DTE connector should be male and DCE connector should be female.
- It can send data at 20kBd for a distance of 50 ft.
- The voltage level for RS-232 are:
  - A logic high or 1 or mark, –3V to –15V
  - A logic low or 0 or space, +3V to +15V
- Normally, ±12V voltage levels are used.

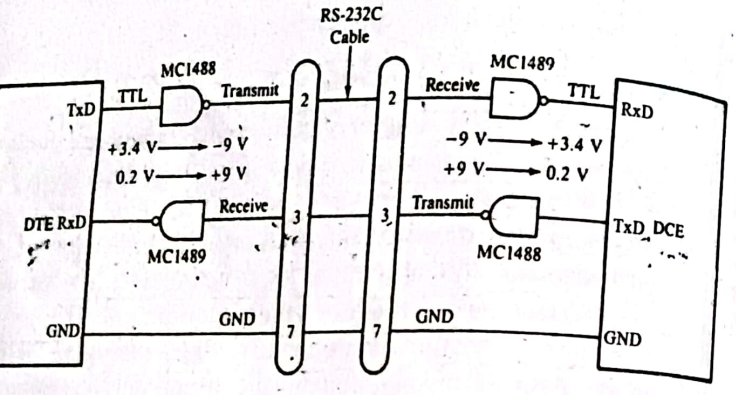

Figure 4.17: Connection of DTE and DCE through RS-232C interface

- MC1488 line driver converts logic 1 to –9V, logic 0 to +9V
- MC1489 line receiver converts RS-232 to TTL
- Signal levels of RS-232 are not compatible with that of the DTE and DCE which are TTL signals for that line driver such as MC1488 and line receiver MC1489 are used.

Table 4.2: RS-232C signals used in handshaking

Signal Flow	DE-9P	DB-25P	Signal	Description
-	-	1	-	Protective Ground
DTE to DCE	3	2	TxD	Transmitted Data
DCE to DTE	2	3	RxD	Received Data
DTE to DCE	7	4	$\overline{RTS}$	Request To Send
DCE to DTE	8	5	$\overline{CTS}$	Clear To Send
DCE to DTE	6	6	$\overline{DSR}$	Data Set Ready
Common Ref	5	7	GND	Signal Ground
DCE to DTE	1	8	$\overline{DCD}$	Data Carrier Detect
DTE to DCE	4	20	$\overline{DTR}$	Data Terminal Ready
DCE to DTE	9	22	RI	Ring Indicator
DCE to DTE	-	23	DSRD	Data Signal Rate Detector

- **Data terminal ready ($\overline{DTR}$):** After the terminal power is turned on and terminal runs any self checks, it asserts data terminal ready ($\overline{DTR}$) signal to tell the modem that it is ready.
- **Data set ready ($\overline{DSR}$):** When the MODEM is powered up and ready to transmit or receive data, it will assert data set ready ($\overline{DSR}$) to the terminal. Under manual control or terminal control, modem then dials up the

computer. If the computer is available, it will send back a specified tone.

- **Request to send ($\overline{RTS}$):** When a terminal has a character ready to send, it will assert a request-to-send ($\overline{RTS}$) signal to the modem.
- **Data carrier detect ($\overline{DCD}$):** The modem will then assert its data-carrier-detect ($\overline{DCD}$) signal to the terminal to indicate that it has established connection with the computer.
- **Clear to send ($\overline{CTS}$):** When the modem is fully ready to receive data, it asserts the clear-to-send ($\overline{CTS}$) signal back to the terminal.
- **Ring indicator (RI):** It indicates that a ring has occurred at modem. Deactivating DTR or DSR breaks the connection but RI works independently of DTR i.e., a modem may activate RI signal even if DTR is not active.
- **Transmitted data (TxD):** The terminal then sends serial data characters to the modem.
- **Received data (RxD):** Modem will receive data from terminal through this line.
- **Data signal rate detect (DSRD):** It is used for switching different baud rate.

2. **RS-423A**

A major problem with RS-232C is that it can only transmit

- will indic...
- During synchronous mode, the signal SYNDET/BRKDET will indicate the reception of synchronous character.

MODEM Control

- The MODEM control unit allows to interface a MODEM to 8251A and to establish data communication through MODEM over telephone lines.
- This unit takes care of handshake signals for MODEM interface.

4.5.4 Universal Serial Bus (USB)

In the past, connecting multiple peripheral devices to computer has been a real problem. There were too many different port types (serial port, parallel port, PS/2 etc.) and their use imposes limitations such as no hot-plug ability and automatic configuration. USB is designed to allow many peripherals to be connected using a single standardized interface. It provides an expandable, fast, bi-directional, low-cost, hot-pluggable Plug and Play serial hardware interface that makes the life of the computer

users easier allowing them to plug different peripheral devices into a USB port and have them automatically configured and ready to use. Using a single connector type, USB allows the user to connect a wide range of peripheral devices, such as keyboards, mice, printers, scanners, mass storage devices, telephones, modems, digital still-image cameras, video cameras, audio devices to a computer. USB devices do not directly consume system resources.

USB is an industry standard developed in the mid-1990s that defines the cables, connectors and protocols used for connection, communication and power supply between computers and electronic devices. It has become commonplace on other devices, such as smart phones, PDAs, and video game consoles. USB has effectively replaced a variety of earlier interfaces, such as serial and parallel ports, as well as separate power chargers for portable devices.

Features of USB

- **Single connector type:** USB replaces all the different legacy connectors with one well-defined, standardized USB connector for all USB peripheral devices, eliminating the need for different cables and connectors and thus simplifying the design of the USB devices. So, all USB devices can be connected directly to a standard USB port on a computer.
- **Hot-swappable:** USB devices can be safely plugged and unplugged as needed while the computer is running. So there is no need to reboot.
- **Plug and play:** Operating system software automatically identifies, configures, and loads the appropriate device driver when a user connects a USB device.
- **High performance:** USB offers low speed (1.5 Mbit/s), full speed (12 Mbit/s) and high speed (up to 480 Mbit/s) transfer rates that can support a variety of USB peripherals. USB 3.0 (super speed USB) achieves the throughput up to 5.0 Gbit/s.

- **Expandability:** ... peripheral devices may theoretically be connected to a single bus at one time.
- **Power supplied from the bus:** USB distributes the power to all connected devices eliminating the need for external power source for low-power devices. High-power devices can still require their own local power supply. USB also supports power saving suspend/resume modes.
- **Easy to use for end user:** A single standard connector type for all USB devices simplifies the end user's task at figuring out which plugs go into which sockets. The operating system automatically recognizes the USB device attachment and loads appropriate device drivers.
- **Low-cost implementation:** Most of the complexity of the USB protocol is handled by the host, which along with low-cost connection for peripherals makes the design simple and low cost.
- **Wide range of workloads and applications:**
  - Suitable for device bandwidths ranging from a few kb/s to several Mb/s
  - Supports isochronous as well as asynchronous transfer types over the same set of wires
  - Supports concurrent operation of many devices (multiple connections)
  - Supports up to 127 physical devices
  - Supports transfer of multiple data and message streams between the host and devices
  - Allows compound devices (i.e., peripherals composed of many functions)
  - Lower protocol overhead, resulting in high bus utilization

Isochronous bandwidth:

- Guaranteed bandwidth and low latencies appropriate for telephony, audio, etc.

- Isochronous workload may use entire bus bandwidth

Robustness:

- Error handling/fault recovery mechanism is built into the protocol.
- Dynamic insertion and removal of devices is identified in user-perceived real-time.
- Supports identification of faulty devices

USB Standards

Like any successful standard, USB (universal serial bus) has kept pace with technology and the standard has been updated with the release of USB 1.0, USB 1.1, USB 2.0, USB 3.0, and then USB 3.1. Each successive USB standard has added more to the technology, improving and refining the performance. With the use of USB being so widespread, backwards compatibility as far as is possible is very important, along with a future upgrade path.

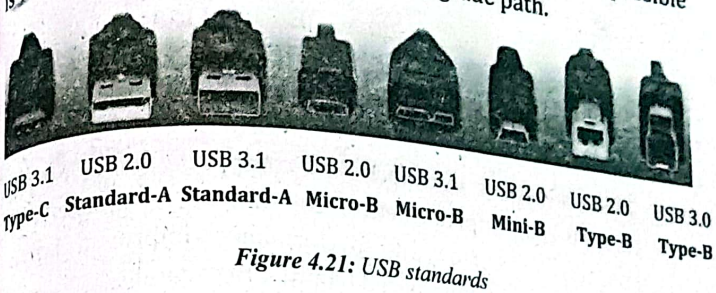

Figure 4.21: USB standards

1. **USB 1.0**

   Released in January 1996, it specified data rates of 1.5 Mbit/s (low bandwidth or low speed) and 12 Mbit/s (full bandwidth or full speed). It did not allow for extension cables or pass-through monitors, due to timing and power limitations. Few such devices actually made it to market.

2. **USB 1.1**

   This was the original version of the USB and was released in August 1998 after a few problems with the USB 1.0 specification were resolved. It provided a master/slave interface and a tiered star topology which was capable of supporting up to 127 devices and a maximum of six tiers or

hubs. The mas... PC with the slaves or "devices" linked via the cable.

The data transfer rates of USB 1.1 are defined as:

Low speed: 1.5 Mbps

Full speed: 12 Mbps

The cable length for USB 1.1 is limited to 5 metres, and the power consumption specification allows each device to take up to 500mA, although this is limited to 100mA during start-up.

3. **USB 2.0**

The USB 2.0 standard also known as hi-speed USB which was released in April 2000. The main difference when compared to USB 1.1 was the data transfer speed increased up to a "high speed" rate of 480 Mbps.

4. **USB 3.0**

This improved USB standard also known as super-speed USB which was released in November 2008. The major feature is the super-speed bus, which provides a fourth transfer mode which gives data transfer rates of 4.8 Gbit/s. This standard is also backward compatible with USB 2.0. USB 3.0 is an improvement of the USB 2.0 technology with an increased speed and performance, improved power management, and increased bandwidth capability.

5. **USB 3.1**

USB 3.1, also known as SuperSpeed+, was made available as of July, 2013, and is the latest version of the USB protocol. USB 3.1 is capable of transfer rates of up to 10 Gbps.

6. **USB 3.2**

USB 3.2, released in September 2017, preserves existing USB 3.1 SuperSpeed and SuperSpeed+ data modes but introduces two new SuperSpeed+ transfer modes over the USB-C connector with data rates of 10 and 20 Gbit/s.

Wirel...

Wireless USB evolved from the universal serial bus concept in 2007 and 2008. It is a high bandwidth radio technology that transmits data from devices like printers, cameras, flash drives as well as streaming video at a fast connection without the use of cables. Wireless USB operates best within a short range. It can transfer data at approximately 110 megabits per second at 30 feet and 480 megabits per second at 10 feet. Wireless USB works to expand a user's options for mobile computing while still maintaining the connectivity of the standard 127 devices.

Chapter 5

Interrupt Operations

5.1 Interrupt and its Importance

5.2 Polling versus Interrupt

5.3 Interrupt Structures

5.3.1 Polled Interrupts

5.3.2 Daisy Chain (Vectored) Interrupt

5.4 Interrupt Processing Sequence

5.5 Multiple Interrupts and Priorities

5.6 Interrupts Types

5.7 Interrupts in 8085

5.8 Interrupts in 8086

5.9 Priority Interrupt Controller (PIC)

INTERRUPT OPERATIONS

5.1 Interrupt and its Importance

Virtually all computers provide a mechanism by which other modules (I/O, memory) may interrupt the normal processing of the processor. *Interrupt* is signal send by an external device to the processor, to request the processor to perform a particular task or work. Table 5.1 lists the most common classes of interrupts.

Table 5.1: Classes of interrupts

Programs	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware Failure	Generated by a failure, such as power failure or memory parity error.

Interrupts are provided primarily as a way to improve processing efficiency. For example, most external devices are much slower than the processor. Suppose that the processor is transferring data to a printer using the instruction cycle we discussed in the previous chapter. After each write operation, the processor must pause and remain idle until the printer catches up. The length of this pause may be on the order of many hundreds or even thousands of instruction cycles that do not involve memory. Clearly, this is a very wasteful use of the processor. So interrupts are particularly useful when interfacing I/O devices that provide or require data at relatively low data transfer rate.

With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress. The interrupt is a process of data transfer whereby an external device or a peripheral can inform the processor that it is ready for communication and it requests attention. The response to an interrupt request is directed or controlled by the microprocessor.

5.2 Polling versus Interrupt

The interrupt process allows the microprocessor to respond to external requests for attention or service on a demand basis and leaves the microprocessor free to perform other tasks. On the other hand, in the polled or status check I/O, the microprocessor remains in a loop doing nothing, until the device is ready for data transfer.

- Each time the device is given a command, for example *"move the read head to sector 42 of the floppy disk"* the device driver has a choice as to how it finds out that the command has completed. The device drivers can either poll the device or they can use interrupts.
- Polling the device usually means reading its status register every so often until the device's status changes to indicate that it has completed the request.
- Polling means the CPU keeps checking a flag to indicate if something happens.
- An interrupt driven device driver is one where the hardware device being controlled will cause a hardware interrupt to occur whenever it needs to be serviced.
- With interrupt, CPU is free to do other things, and when something happens, an interrupt is generated to notify the CPU. So it means the CPU does not need to check the flag.
- Polling is like picking up your phone every few seconds to see if you have a call. Interrupts are like waiting for the phone to ring.
- Interrupts win if processor has other work to do and event response time is not critical.

- Polling can be better if processor has to respond to an event ASAP; may be used in device controller that contains dedicated secondary processor.

Advantages of interrupt over polling:

- Interrupts are used when we need the fastest response to an event. For example, we need to generate a series of pulses using a timer. The timer generates an interrupt when it overflows and within 1 or 2 sec, the interrupt service routine is called to generate the pulse. If polling were used, the delay would depend on how often the polling is done and could delay response to several msecs. This is thousands times slower.
- Interrupts are used to save power consumption. In many battery powered applications, the microcontroller is put to sleep by stopping all the clocks and reducing power consumption to a few micro amps. Interrupts will awaken the controller from sleep to consume power only when needed. Applications of this are hand held devices such as TV/VCR remote controllers.
- Interrupts can be a far more efficient way to code.
- Interrupts are used for program debugging.

5.3 Interrupt Structures

A processor is usually provided with one or more interrupt pins on the chip. Therefore a special mechanism is necessary to handle interrupts from several devices that share one of these interrupt lines. There are mainly two ways of servicing interrupts: *polled interrupts* and *daisy chain (vectored) interrupts*.

5.3.1 Polled Interrupts

Polled interrupts are handled by using software, and is slower than hardware interrupts. Here the processor has the general (common) interrupt service routine (ISR) for all devices. The priority of the devices is determined by the order in which the routine polls each device. The processor checks the starting with

the highest priority device. Once it determines the source of the interrupt, it branches to the service routine for that device.

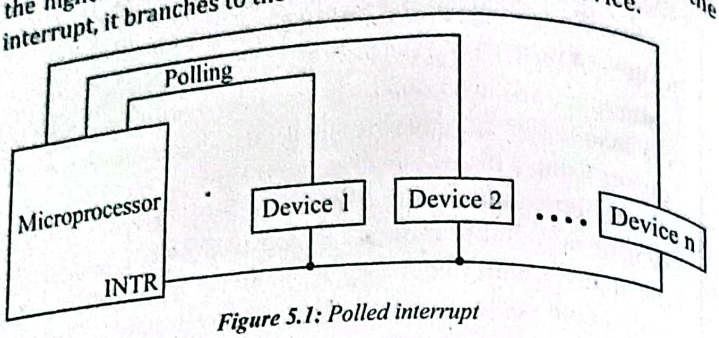

Figure 5.1: Polled interrupt

Here, several external devices are connected to a single interrupt line (INTR) of the microprocessor. When INTR signal goes up, the processor saves the contents of PC and other registers and then branches to an address defined by the manufactures of the processor. The user can write a program at this address to find the source of the interrupt by starting the polled from highest priority device.

5.3.2 Daisy Chain (Vectored) Interrupt

In polled interrupt, the time required to poll each device may exceed the time to service the device through software. To improve this, the faster mechanism called *vectored* or *daisy chain interrupt* is used. Here, the devices are connected in chain fashion. This configuration is governed by the priority of the devices. The device with the highest priority is placed first followed by the second highest priority device and so on. When INTR pin goes up, the processor saves its current status and then generates $\overline{\text{INTA}}$ signal to the highest priority device. If this device has generated the interrupt, it will accept the INTA; otherwise it will push $\overline{\text{INTA}}$ to the next priority device until the $\overline{\text{INTA}}$ is accepted by the interrupting device.

When $\overline{\text{INTA}}$ is accepted, the device provides a means to the processor for findings the interrupt address vector using external hardware. The accepted device responds by placing a word on the data lines which becomes the vector address with the help of any

hardware ... processor points to appropriate device service routine. Here no general interrupt service routine need first that means appropriate ISR of the device will be called.

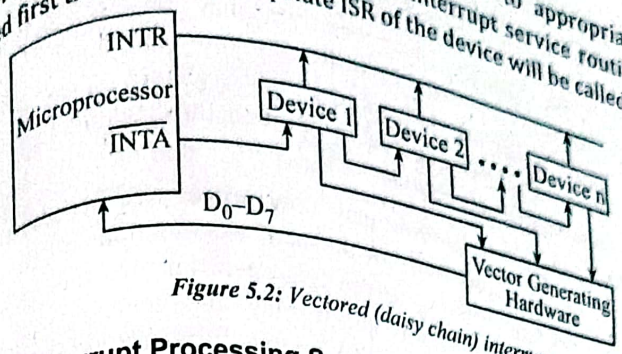

Figure 5.2: Vectored (daisy chain) interrupt

5.4 Interrupt Processing Sequence

The occurrence of an interrupt triggers a number of events, both in the processor hardware and in software. The interrupt driven I/O operation takes the following steps:

- The I/O unit issues an interrupt signal to the processor for exchange of data between them.
- The processor finishes execution of the current instruction before responding to the interrupt.
- The processor tests for an interrupt, determines that there is one, and sends an acknowledgement signal to the device that issued the interrupt.
- The processor now needs to prepare to transfer control to the requested routine called *interrupt service routine* (ISR). To begin, it needs to save information needed to resume the current program at the point of interrupt. The minimum information required is the *program status word* (PSW) and the location of the next instruction be executed, which is contained in the *program counter* (PC). The contents of PSW and PC are pushed onto the stack.
- The processor loads the program counter with the entry location of the *interrupt-handler program* that will respond to this interrupt, that is, the control is transferred to the interrupt-handler program.

2. **Prioritywise processing of interrupts**

The drawback of sequential processing is that it does not take account of relative priority or time critical needs. The alternative form of this is to define priorities for interrupts and to allow an interrupt of higher priority to cause a lower priority interrupts pause until high priority interrupt completes its function.

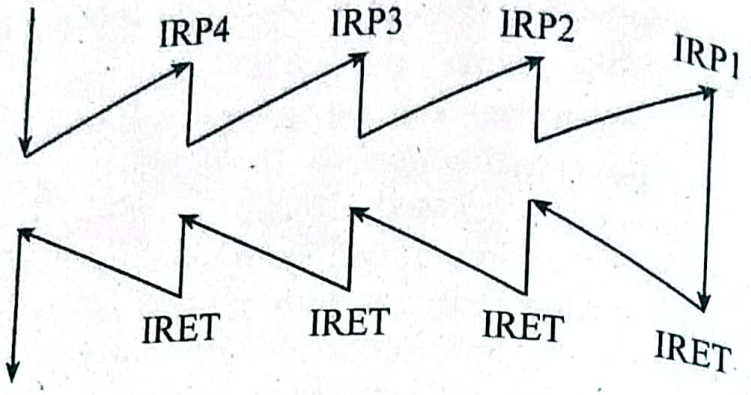

Figure 5.5: Prioritywise interrupt service

Interrupt Service Routine

An *interrupt service routine* (ISR) is a software routine that hardware invokes in response to an interrupt. ISRs examine an interrupt and determine how to handle it. ISRs handle the interrupt, and then return a logical interrupt value. Its central purpose is to process the interrupt and then return control to the main program. An ISR must perform very fast to avoid slowing down the operation of the device and the operation of all lower priority ISRs. As in procedures, the last instruction in an ISR should be IRET.

ISR is responsible for doing the following things:

1. **Saving the processor context**

   Because the ISR and main program use the same processor registers, it is the responsibility of the ISR to save the processor's registers before beginning any processing of the interrupt. The processor context consists of the instruction pointer, registers, and any flags. Some processors perform this step automatically.

Non-maskable interrupts are those which cannot be enabled or disabled by microprocessor by executing instructions. TRAP is a non-maskable interrupt. It consists of both level as well as edge triggering and is used in critical power failure conditions.

3. **Vectored and Non-Vectored Interrupts**

Vectored interrupts are those which have fixed vector address (starting address of sub-routine) and after executing these, program control is transferred to that address.

Non-vectored interrupts are those in which vector address is not predefined. The interrupting device gives the address of sub-routine for these interrupts. INTR is the non-vectored interrupt.

4. **External and Internal Interrupts**

An *external interrupt* is a computer system interrupt that happens as a result of outside interference, whether that's from the user, from peripherals, from other hardware devices or through a network.

An *internal interrupt* is a specific type of interrupt that is caused by instructions embedded in the execution instructions of a program or process. Typically, internal interrupts resist changes by users, and happen "naturally" or "automatically" as a processor works through program instructions, rather than being caused by external events or network connections. These are indicated internally by exceptional conditions such as overflow, divide by zero, and execution of illegal op-code. The user usually writes a service routine to take correction measures and to provide an indication in order to inform the user that exceptional condition has occurred.

5.7 Interrupts in 8085

The 8085 interrupt process can be described as follows:

1. The interrupt process should be enabled using instruction EI which sets the interrupt enable flip-flop. The instruction DI resets the flip-flop and disables the interrupt process.

2. When the microprocessor is executing a program, it checks the INTR line during the execution of each instruction at the 2nd T-state of last machine cycle.

3. If the line INTR is high and the interrupt is enabled, the microprocessor completes the current instruction, disables the interrupt enable flip-flop, and sends an interrupt acknowledge signal ($\overline{INTA}$).

4. The program control is transferred to the interrupt service routine (ISR). Now the processor saves the memory address of the next instruction and the contents of registers on the stack.

5. The processor services the interrupt using interrupt service routine. At the end of the subroutine, the RET instruction retrieves the memory address where the program was interrupted and continues the execution.

Types of 8085 Interrupts

The 8085 microprocessor has multilevel interrupt system. It supports two types of interrupts: *hardware interrupts* and *software interrupts*.

1. **Software Interrupts**

8085 allows some special instructions to interrupt the main program. After executing these instructions, 8085 completes the execution of instruction it is currently executing and transfers the program control to an interrupt service routine. Upon completion of the service routine, execution returns to the main program. These types of interrupts where instructions cause interrupt requests are known as software interrupts.

2. **Hardware Interrupts**

Some PINs on the 8085 allow peripheral devices to interrupt the main program for I/O operations. When an interrupt occurs, 8085 completes the execution of instruction it is currently executing and transfers the program control to an interrupt service routine that services the peripheral devices. Upon completion of the

service routine; execution returns to the main program. These types of interrupts where MPU PINs are used to receive interrupt requests are known as hardware interrupts.

Software Interrupts

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

Interrupt number × 8 = vector address

For example, 5 × 8 = 40 = 28H

Vector address for interrupt RST 5.5 is 0028H

Table 5.9 shows the vector addresses of all interrupts.

Table 5.2: Vector addresses for software interrupts

Instruction	Hexcode	Vector Address
RST 0	C7H	0000H
RST 1	CFH	0008H
RST 2	D7H	0010H
RST 3	DFH	0018H
RST 4	E7H	0020H
RST 5	EFH	0028H
RST 6	F7H	0030H
RST 7	FFH	0038H

Hardware Interrupts

The 8085 has five hardware interrupts.

Table 5.3: 8085 hardware interrupts

S.N.	Interrupt	Trigger	Priority	Maskable	Vector Address
1	TRAP	Edge and Level	1st	No	0024H
2	RST 7.5	Edge	2nd	Yes	003CH
3	RST 6.5	Level	3rd	Yes	0034H
4	RST 5.5	Level	4th	Yes	002CH
5	INTR	Level	5th	Yes	-

When any of these PINs except INTR is active, the internal control circuit of 8085 produces a CALL to a predetermined memory location. This memory location, where the service routine starts is referred to as vectored location and such interrupts are known as vectored interrupts. The INTR is not a vectored interrupt, it receives the address of ISR from the external device.

In 8085, all interrupts except TRAP are maskable. When logic signal is applied to a maskable interrupt input, the 8085 is interrupted only if that particular input is enabled. These interrupts can be enabled or disabled under program control. If disabled, 8085 disables an interrupt request. The interrupt TRAP is non-maskable that means it is under program control.

TRAP

This interrupt is a non-maskable vectored interrupt, it is unaffected by any mask or interrupt enable. TRAP has the highest priority and is edge and level triggered. This means that the TRAP must go high and remain high until it is acknowledged. It cannot be acknowledged again until it makes a transition from high to low to high. This avoids false triggering caused by noise and transients. When this interrupt is triggered, the program control is transferred to location 0024H without an external hardware or interrupt enable instruction EI. TRAP is generally used for critical events such as sudden power failure and emergency shut-off

There are two ways to clear TRAP interrupt.

1. By resetting microprocessor that is by giving low signa on $\overline{\text{RESETIN}}$ (external signal).
2. By giving a high TRAP acknowledge (internal signal).

RST 7.5

The RST 7.5 interrupt is a maskable vectored interrupt. It has the second highest priority. It is positive edge sensitive and can be triggered with a short pulse that is input goes to high and no need to maintain high state until it recognized. It can be enabled with two instructions EI (Enable Interrupt) and SIM (Set Interrupt Mask). When this interrupt is triggered, the program control is transferred to location 003CH.

RST 7.5 can be disabled by:

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

RST 6.5 and RST 5.5

The RST 6.5 and RST 5.5 both are maskable vectored interrupts. The RST 6.5 has the third priority whereas the RST 5.5 has the fourth priority. These are level triggered that is input goes to high and stay high until it is recognized. It can be enabled with two instructions EI (Enable Interrupt) and SIM (Set Interrupt Mask). When RST 6.5 interrupt is triggered, the program control is transferred to location 0034H and when RST 6.5 interrupt is triggered, the program control is transferred to location 002CH.

They can be disabled by

1. By using DI or SIM instructions.
2. System or processor reset.
3. After reorganization of interrupt.

NTR

The INTR is a maskable non-vectored interrupt. It has the west priority. It can be enabled by EI (Enable Interrupt) struction. It is a level triggered interrupt that is input goes to gh and it is necessary to maintain high state until it recognized. ter receiving $\overline{\text{INTA}}$ signal, it has to supply the address of ISR.

It can be disabled by

1. By using DI or SIM instructions.
2. System or processor reset.

3. After reorganization of interrupt.

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

1. The 8085 checks the status of INTR signal during execution of each instruction.
2. If INTR signal is high, then 8085 complete its current instruction and sends active low interrupt acknowledge signal $\overline{\text{INTA}}$, if the interrupt is enabled.
3. In response to $\overline{\text{INTA}}$ signal, external logic places an instruction opcode on the data bus. In the case of multi-byte instruction, additional interrupt acknowledge machine cycles are generated by the 8085 to transfer the additional bytes into the microprocessor.
4. On receiving the instruction, the 8085 save the address of next instruction on stack and execute received instruction.

8085 Interrupt Instructions

Maskable interrupts are enabled or disabled that is masked or unmasked using program control. There are four instructions used for program control in 8085 microprocessor namely EI, DI, SIM and RIM.

1. **EI (Enable Interrupt)**

   The EI instruction sets the interrupt enable flip-flop. Thus RST 7.5, RST 6.5, RST 5.5 and INTR are enabled using EI instruction. When any interrupt is acknowledged or system reset, interrupt enable flip-flop resets and disables all interrupts. To enable interrupt in further process it is necessary to execute EI instruction within interrupt service routine. It is 1-byte instruction and no flags are affected.

2. **DI (Disable Interrupt)**

   DI instruction resets interrupt enable flip-flop. Thus it disables RST 7.5, RST 6.5, RST 5.5 and INTR interrupts. It is of 1-byte instruction and no flags are affected.

3. **SIM (Set Interrupt Mask)**

This instruction is used to set interrupt mask and to send serial output. It transfers the content of accumulator to interrupt control logic and serial I/O ports. Thus it is necessary to load appropriate contents in the accumulator before execution of SIM instruction. It is of 1-byte instruction and no flags are affected. The format of SIM instruction is shown in figure below.

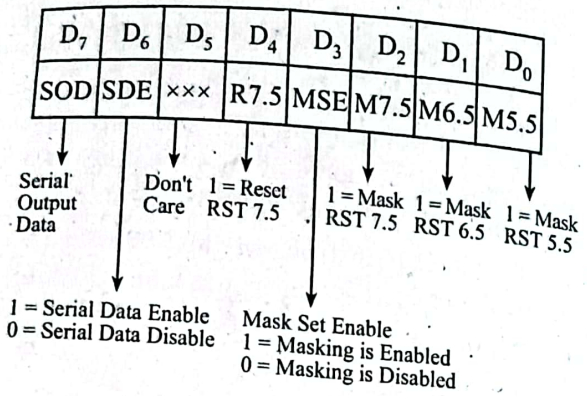

Figure 5.11: SIM instruction format

Actually, SIM does the following three tasks:

1. Masks the interrupts, bits 0 to bit 2 set/reset the mask bits in Interrupt mask register. Bit 0 masks RST 5.5, bit 1 masks RST 6.5 and bit 3 masks RST 7.5. Bit 3 enables/disables the masking control of bits 0 – 2.
2. Resets RST 7.5, bit 4 is used to reset it whether RST 7.5 is masked or not. This is mainly used to overwrite RST 7.5 without serving it.
3. Implements serial I/O. If bit 6 = 1 is used to enable serial I/O and bit 7 is used to transmit serial output data bit.

RIM (Read Interrupt Mask)

The RIM instruction loads the status of the interrupt mask, the pending interrupts and the contents of serial input data line, SID, into accumulator. Thus it is possible to monitor the status of interrupt mask, pending interrupts and serial input. The format of RIM instruction is shown in figure below.

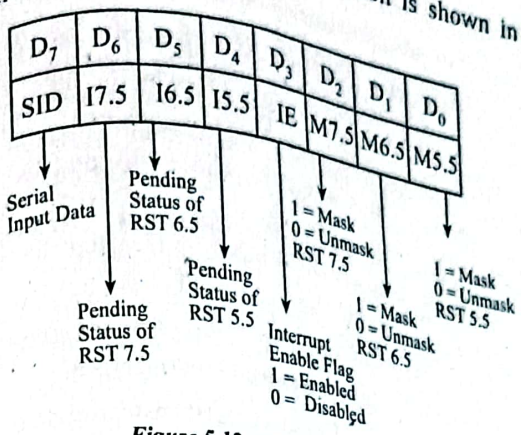

Figure 5.12: RIM instruction format

Actually, RIM does the following four tasks:

1. Checks whether RST 5.5, RST 6.5, and RST 7.5 are masked or not; bit 0 to bit 2 show the current setting of the mask status of interrupts RST 5.5, RST 6.5 and RST 7.5. logic 1 indicates the interrupt is masked.
2. Checks whether interrupts are enabled or not; bit 3 shows the status of interrupt enable flip flop. Logic 1 indicates that the interrupts are enabled.
3. Checks whether RST 5.5, RST 6.5, and RST 7.5 interrupts are pending or not; bit 4 to bit 6 show the status of pending interrupts RST 5.5, RST 6.5, and RST 7.5. Logic 1 indicates the interrupt is pending.
4. Performs serial input of data; bit 7 is used for Serial Data Input.

5.8 Interrupts in 8086

An 8086 interrupt can come from any one of three sources. One is an external signal applied to NMI pin or to INTR pin, and are referred to as *hardware interrupts*. Another source of an interrupt is execution of the interrupt instruction and is referred

to as *software interrupt*. The third source of interrupt is some error conditions produced during the execution of an instruction.

The 8086 interrupt process can be described as follows:

1. At the end of each instruction cycle, the 8086 checks to see if any interrupts have been requested.
2. If there is any interrupt, the 8086 pushes the flag register on the stack.
3. The 8086 disables INTR input by clearing the interrupt flag (IF) and resets the trap flag (TF).
4. The contents of current code segment register and instruction pointer are pushed on the stack.
5. Then, the program control is transferred to interrupt service routine. An IRET instruction at the end of ISR returns execution to the main program by retrieving the contents which were pushed on the stack before.

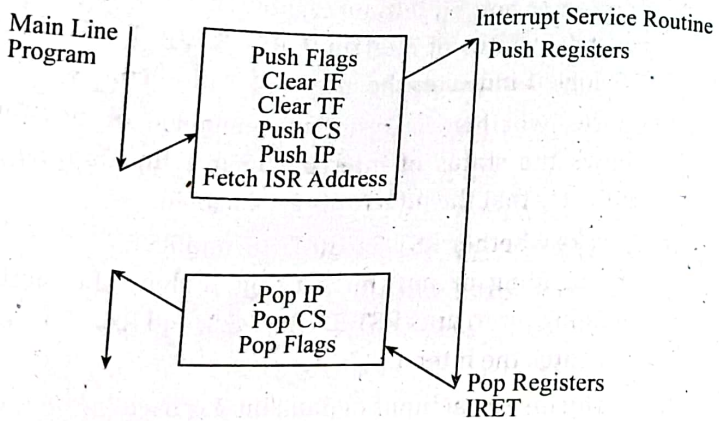

Figure 5.13: 8086 interrupt response

8086 Interrupt Types

There are two types of Interrupts in 8086. They are:

- **Hardware Interrupts (External Interrupts)**

Hardware interrupts are those interrupts which are caused by any peripheral device by sending a signal through a specified pin to the microprocessor. There are two hardware interrupts in 8086 microprocessor:

i. **NMI (Non-Maskable Interrupt):** It is a single pin non-maskable hardware interrupt which cannot be disabled. It is the highest priority interrupt in 8086 microprocessor. After its execution, this interrupt generates a TYPE 2 interrupt. IP is loaded from word location 00008H and CS is loaded from the word location 0000AH.

ii. **INTR (Interrupt Request):** It provides a single interrupt request and is activated by I/O port. This interrupt can be masked or delayed. It is a level triggered interrupt. It can receive any interrupt type, so the value of IP and CS will change on the interrupt type received. The INTA is the response pin for INTR request.

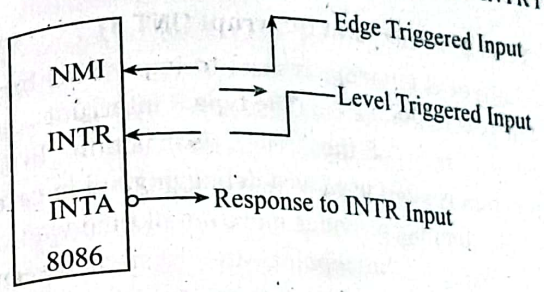

Figure 5.14: 8086 hardware interrupts

2. **Software Interrupts (Internal Interrupts)**

These are instructions that are inserted within the program to generate interrupts. There are 256 software interrupts in 8086 microprocessor. The instructions are of the format INT type where type ranges from 00H to FFH. The starting address ranges from 00000H to 003FFH. These are 2 byte instructions. IP is loaded from (type × 04H) and CS is loaded from the next address given by (type × 04H + 02H). Some important software interrupts are:

- **Type 0 : Divide by Zero Interrupt**

When the quotient from either a DIV or IDIV instruction is too large to fit in the result register; 8086 Interrupt Types will automatically execute type 0 interrupt.

- **Type 1 : Single Step Interrupt**

  The type 1 interrupt is the single step trap. In the single step mode, system will execute one instruction and wait for further direction from user. Then user can examine the contents of registers and memory locations and if they are correct, user can tell the system to execute the next instruction. This feature is useful for debugging assembly language programs.

- **Type 2 : Non-Maskable Interrupt (NMI)**

  As the name suggests, this interrupt cannot be disabled by any software instruction. This interrupt is activated by low to high transition on 8086 NMI input pin. In response, 8086 will do a type 2 interrupt.

- **Type 3 : Breakpoint Interrupt (INT 3)**

  The type 3 interrupt is used to implement break point function in the system. The type 3 interrupt is produced by execution of the INT 3 instruction. Break point function is often used as a debugging aid in cases where single stepping provides more detail than wanted. When we insert a breakpoint, the system executes the instructions up to the breakpoint, and then goes to the breakpoint procedure. In the break point procedure we can write a program to display register contents, memory contents and other information that is required to debug our program. We can insert as many breakpoints as we want in our program.

- **Type 4 : Overflow Interrupt (INT O)**

  The type 4 interrupt is used to check overflow condition after any signed arithmetic operation in the system. The 8086 overflow flag (OF), will be set if the signed result of an arithmetic operation on two signed numbers is too large to be represented in the destination register or memory location.

We can prioritize the 8086 interrupts as follows:

Table 5.4: 8086 interrupt priority

Interrupt	Priority
Divide Error, INT(n), INTO	Highest
NMI	↓
INTR	
Single Step	Lowest

Interrupt Vector Table (IVT)

The *interrupt vector table* is a feature of the Intel 8086 family of microprocessors. An *interrupt vector* is a 4-byte number stored in the first 1024 bytes of the memory (00000H – 003FFH). There are 256 different interrupt vectors. Each vector contains the address of an interrupt service routine. Each vector contains a value for IP and CS that forms the address of the interrupt service routine. The first two bytes contain the IP, and the last two bytes contain the CS. Thus, *interrupt vector table* (IVT) is a 1024 bytes sized table that contains the addresses of interrupt service routine. The purpose of the IVT is to hold the vectors that redirect the microprocessor to the right place when an interrupt arrives.

The figure below shows the organization of interrupt vector table. The interrupt number is used as an index into the table to get the address of the interrupt service routine. When the 8086 responds to a particular type interrupt, it automatically multiplies the type by 4 to produce the desired address in the interrupt vector table. It then goes to that address in the table to get the starting address of the interrupt service routine.

The lowest five types (type 0 to type 4) are dedicated to specific interrupts such as the divide-by-zero interrupt, the single-step interrupt, the non-maskable interrupt, the breakpoint interrupt, and the overflow interrupt. Interrupt types 5 to 31 are reserved by Intel for use in more complex microprocessors. The upper 224 interrupt types from 32 to 255 are available for us to use for hardware or software interrupts.

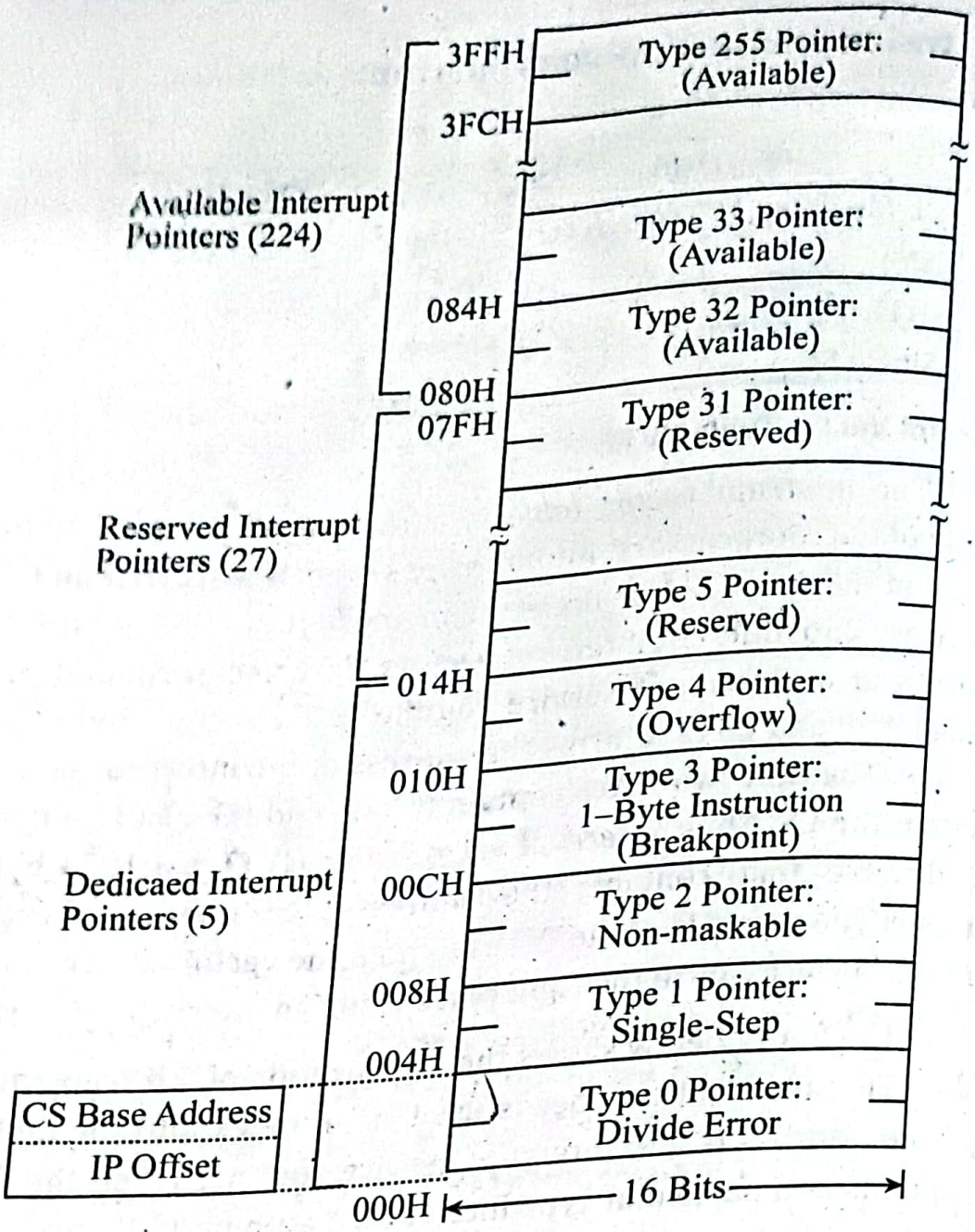

Figure 5.15: IVT structure (organization)

DOS & BIOS interrupts:

DOS interrupts services link applications with os services such as opening file, reading, writing content using certain functions of INT 4H. BIOS interrupts control the screen disk controller and keyboard operation using INT 10H, 13H,16H etc.

9 Priority Interrupt Controller (PIC)

The INTR pin can be used for multiple peripherals, and to determine priorities among these devices when two or more

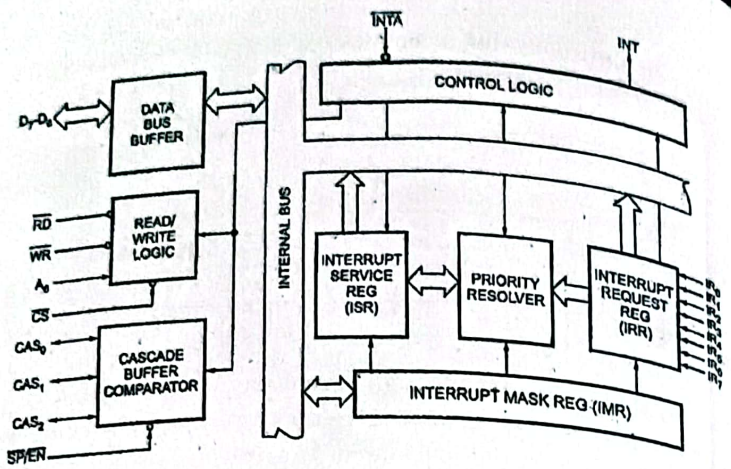

Figure 5.17: Intel 8259A programmable interrupt controller

Data Bus Buffer

This bidirectional 8-bit buffer is used to interface the 8259A to the microprocessor data bus.

Read/Write & Control Logic

It accepts output commands sent from the microprocessor. It contains the initialization command word (ICW) registers and operation command word (OCW) registers which store the various control formats for device operation. It also allows the status of 8259A to be transferred to the data bus.

Interrupt Request Register (IRR)

It stores all the interrupt inputs that are requesting service. It is an 8-bit register; one bit for each interrupt request. Basically, it keeps track of which interrupt inputs are asking for service. If an interrupt input is unmasked, and has an interrupt signal on it, then the corresponding bit in the IRR will be set. The content of this register can be read to know the status of pending interrupts.

Interrupt Mask Register (IMR)

The *interrupt mask register* is used to disable (mask) or enable (unmask) individual interrupt request inputs. This is also

an 8-bit register. Each bit in this register corresponds to the interrupt input with the same number. The IMR operates on the IRR. Masking of higher priority input will not affect the interrupt request lines of lower priority. To unmask any interrupt, the corresponding bit is set '0'.

Interrupt Service Register (ISR)

The *interrupt service register* keeps track of which interrupt inputs are currently being serviced. In 8259A, during the service of an interrupt request, if another higher priority interrupt becomes active, it will be acknowledged and the control will be transferred from lower priority interrupt service subroutine (ISS) to higher priority ISS. Thus, more than one bit of ISR will be set indicating the number of interrupts being serviced.

Priority Resolver

Priority resolver determines the priorities of the interrupts set in the IRR. It takes the information from IRR, IMR and ISR to determine whether the new interrupt request is having highest priority or not. If the new interrupt request is having the highest priority, it is selected and processed.

Cascade Buffer/Comparator

It stores and compares the IDs of all 8259A's in the system. The associated three I/O lines (CAS_2-CAS_0) are outputs when 8259A is used as a master and are inputs when 8259A is used as a slave.

Chapter 6

Advanced Topics

6.1 Harvard Architecture

6.2 Accumulator-Based and Register-Based Architecture

6.3 Hardwired and Microprogrammed Control Unit

6.3.1 Control Unit

6.4 RISC and CISC Architectures

6.4.1 CISC Architecture

6.4.2 RISC Architecture

6.5 Introduction to Multitasking and Multiprocessing

6.6 Digital Signal Processor (DSP)

6.1 Harvard Architecture

In *Von Neumann architecture*, the same memory is used for storing instructions and data. Similarly, a single bus called data bus or address bus is used for reading data and instructions from or writing to memory. This architecture limits the processing speed of computers.

The *Harvard architecture* based computer consists of separate memory spaces for the programs (instructions) and data. Each space has its own address and data buses. So, instructions and data can be fetched from memory concurrently and provides significance processing speed improvement.

In Figure 6.1, there are two data and two address buses multiplexed for data bus and address bus. Hence, there are two blocks of RAM chips: one for program memory and another for data memory addresses.

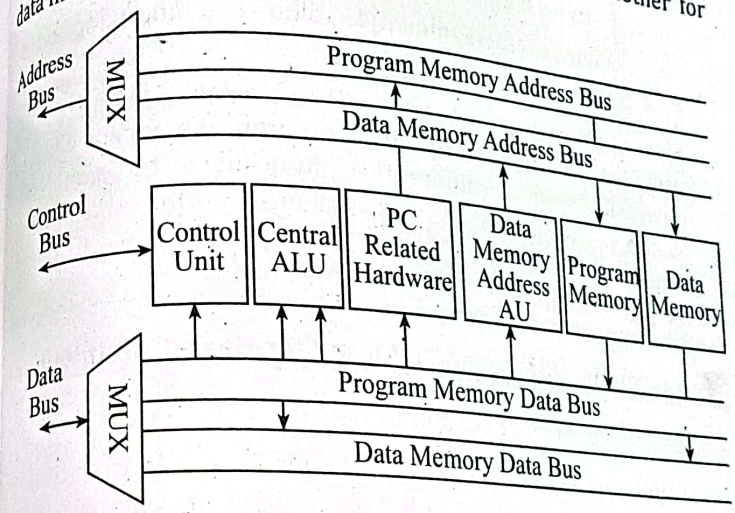

Figure 6.1: *Harvard architecture*

The control unit controls the sequence of operations. Central ALU consists of ALU, multiplier, accumulator, and scaling chief register. The PC is used to address program memory and

always contains the address of next instruction to be executed. Here, data and control buses are bidirectional and address bus is unidirectional.

Differences Between Harvard and Von Neumann Architecture

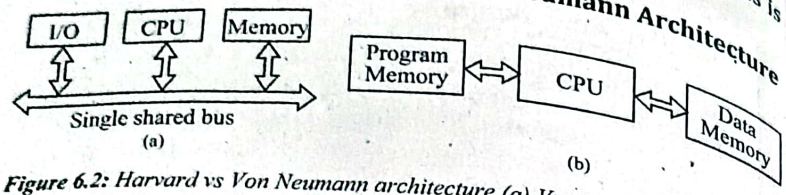

Figure 6.2: Harvard vs Von Neumann architecture (a) Von Neumann architecture (b) Harvard architecture

The differences between Harvard and Von Neumann architecture are:

Harvard architecture	Von Neumann architecture
1. Separate buses for instruction and data fetching	1. Single shared bus for instruction and data fetching
2. Easier to pipeline, so high performance can be achieved	2. Low performance compared to Harvard architecture
3. Comparatively high cost	3. Cheaper
4. No memory alignment problems.	4. Allows self modifying codes
5. Since data memory and program memory are stored physically in different locations, no changes for accidental corruption of program memory.	5. Since data memory and program memory are stored physically in the same chip, chances for accidental corruption of program memory.

6.2 Accumulator-Based and Register-Based Architecture

Accumulator-Based Architecture

In *accumulator-based microprocessor architecture*, accumulator is the most significant register in comparison to other registers and most of the arithmetic and logical operations are performed through accumulator. Data can only enter into the ALU from accumulator and the out pot of the ALU can be stared in accumulator through data bus. The 8085 microprocessor is an example of accumulator-based architecture.

Register-Based Architecture

Unlike an accumulator-based microprocessor architecture, a *register-based microprocessor architecture* has one or more general purpose registers. Data can enter into the ALU from any registers. The 8086 microprocessor is an example of register-based architecture.

The advantages of register-based architecture is extendibility and flexibility in programming. However, the disadvantage is requirement of complex circuitry.

6.3 Hardwired and Microprogrammed Control Unit

6.3.1 Control Unit

The *control unit* is a portion of the processor. The control unit issues control signals external to the processor to cause data exchange with memory and I/O devices. The control unit is also responsible for issuing control signals internal to the processor to move data between registers, to cause the ALU to perform a specified function, and to regulate other internal operations.

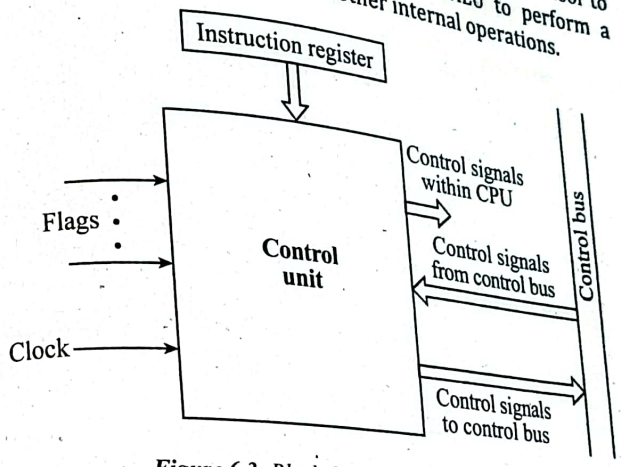

Figure 6.3: Block diagram of the control unit

Source: Computer Organization and Architecture, William Stallings, 8th ed., 2010)

The control unit of a processor performs two tasks:

1. It causes the processor to step through a series of micr operations in the proper sequence, based on the progra being executed; and

2. - It generates the control signals that cause each micro-operation to be executed.

Implementation of control unit fall into one of two categories:

1. Hardwired implementation
2. Microprogrammed implementation

1. Hardwired Control Unit

Hardwired control unit is a combinational logic circuit. In hardwired control unit, for each control signal, Boolean expression has to be derived for that signal as a function of the inputs. Thus, there would be a set of Boolean equations that define the behaviour of the control unit. In a modern complex processor, for defining the control unit, the number of Boolean equations required is very large. As a result, implementing a combinational circuit that satisfies all of these equations becomes extremely difficult. Due to this reason, a simpler approach known as *microprogrammed control unit* is used.

Hardwired control unit has faster mode of operation. A hardwired control unit needs rewiring if design has to be modified.

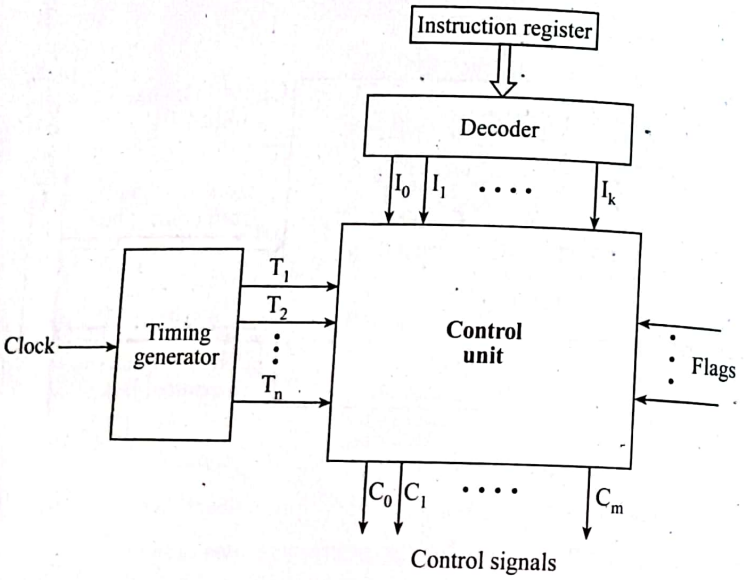

Figure 6.4: Hardwired control unit

Source: Computer Organization and Architecture, William Stallings, 8th ed., 2010)

2. Microprogrammed Control Unit

In a *microprogrammed control unit*, the logic of the control unit is specified by a microprogram. A *microprogram* is a sequence of instructions that specify micro-operations.

A microprogrammed control unit is a relatively simple logic circuit that is capable of (i) sequencing through microinstructions and (ii) generating control signals to execute each microinstruction.

Modifications in microprogrammed control unit can be done by changing the microinstructions.

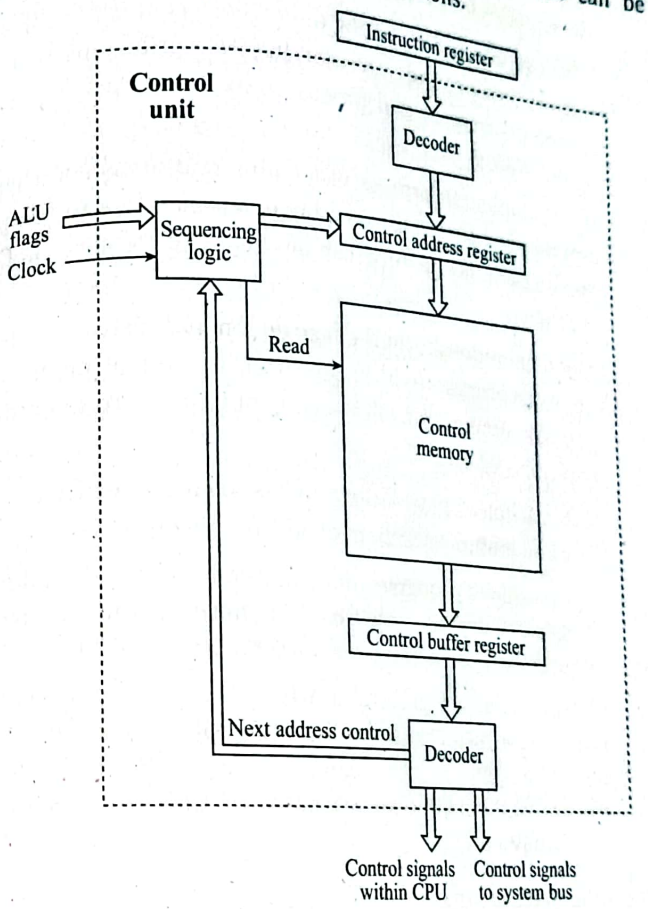

Figure 6.5: Microprogrammed control unit

Source: Computer Organization and Architecture, William Stallings, 8th ed., 2010

6.4 RISC and CISC Architectures

6.4.1 CISC Architecture

Complex instruction set computers is the acronym for CISC and machine based on this architecture have complex instructions. Most of the personal computers which are used today are based on CISC architecture. Following are the characteristics of CISC machines:

- CISC machines have complex instructions which need a large cycles for execution.
- The transfer of data among the register is much faster than among memory and processor. In CISC, various instructions based on memory and processer so the processing speed gets reduced.
- Pipelining is the process of fetching one instruction when another instruction is executing in parallel. Due to complex instruction this feature cannot be heavily used in CISC machines.
- Micro-operations form the instruction and instruction form he micro-program which is written in control memory to perform timing and sequencing of the micro-operations implemented in CISC.
- CISC machines have large number of complex instructions based on multiple numbers of addressing modes.
- CISC machines processer does not consist of large number of registers due to large cost. So these machines have to perform various memory read and write operations.
- CISC machines are preferable where the speed of processer is not the prime issue and where general applications are to be handled. Processers like 8085, 8086, 8086,8086,8086,8086 etc are based on CISC processers and even today's pc.

4.2 RISC Architecture

The term *RISC* represents reduced instruction set mputing/ computers. It focuses on a small set of instructions

RISC	CISC
3. Heavily pipelined, increasing performance.	3. Not/less pipelined
4. Multiple register sets.	4. Single register set
5. Instructions have fixed format.	5. Instructions have variable format.
6. Few number of instructions and addressing modes.	6. Large number of instructions and addressing modes.
7. Complexity is in the design of compiler.	7. Complexity is in microprogram.
8. Instructions are executed by hardware.	8. Instructions in CISC machine are interpreted by microprogram.
9. Since RISC processors are simpler than CISC processors, they can be designed more quickly.	9. CISC processors have comparatively longer design cycle.
10. Examples include SPARC, MIPS R4000, etc.	10. Examples include IBM 370/168, VAX 11/780, 8085, 8086, 80286, 80386, etc.

5 Introduction to Multitasking and Multiprocessing

Multiprogramming is the management of multiple processes hin a uniprocessor system. It is one of the most important ects of operating systems. With multiprogramming, only one ess can execute at a time; meanwhile all other processes are ing for the processor. *Multiprogrammed system* provides an onment in which the various system resources (CPU, ory, and peripheral devices) are utilized effectively, but it not facilitiate user interaction with the computer system.

Time-sharing or *multitasking system* is a logical extension of rogrammed system in which the CPU executes multiple ses by switching among them, but the switches occur so ntly that the users can interact with each program while it ng.

Multiprocessing is the management of multiple processes within a multiprocessor system. With multiprocessing, more than one process can be running simultaneously, each on a different processor. Multiprocessing is an example of parallelism which uses multiple CPUs sharing the common resources such as memory, storage device etc. *Multiprocessor systems* (also known as *parallel systems* or *multicore systems*) have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices. Nowadays, multiprocessor systems are found in servers, desktop systems, laptop systems, and mobile devices (smartphones, tablet computers).

Characteristics of a multiprocessor system:

- Contains two or more similar general purpose processors of comparable capability.
- All processors share access to common memory.
- All processors share access to I/O devices either through the same channels that provide paths to the same devices.
- System is controlled by an integrated operating system that provides interaction between processors and their programs.

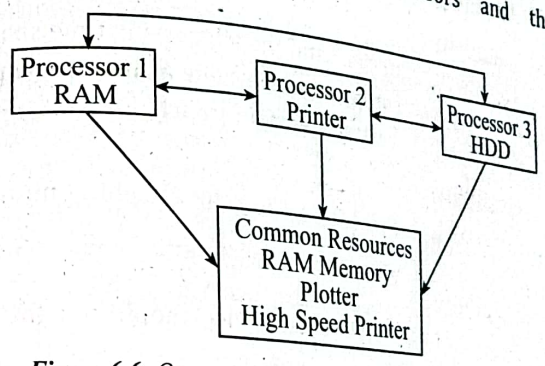

Figure 6.6: *Organization of a multiprocessor system*

Here, the processors can communicate with each other through memory. The CPUs can directly exchange signals as indicated by dotted line.

The organization of a multiprocessor system can be divided into three types:

1. **Time Shared or Common Bus**

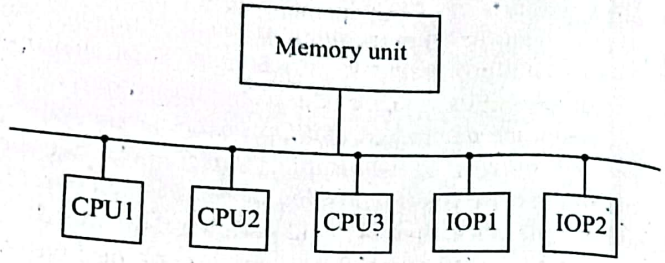

Figure 6.7: Time shared system

There are number of CPUs, I/O modules and memory modules connected to the same bus. So the time shared system must distinguish the modules on the bus to determine the source and destination of the data. Any module in the bus can temporarily act as a master. When one module is controlling the bus, the other should be locked out. The access to each module is divided on the basis of time. The time shared multiprocessing system has the following advantages and drawbacks.

Advantages:

Simplicity

The physical interface and the addressing time sharing logic of each processor remains the same as in a single processor system, so it is very simplest approach.

Flexibility

It is easy to expand the system by attaching more CPUs to the bus.

Reliability

The failure of any attached device should not the failure of the whole system.

Drawbacks:

The speed of the system is limited by the cycle time because all memory references must pass through the common bus.

2. **Multiport Memory**

Each processor and I/O module has dedicated path to each memory module this system has more performance and

complex... portions of memory as private to one or more configure ... possible to CPUs and/or I/O modules. This feature allows increasing security against unauthorized access, and the storage of recovery routines in areas of memory not susceptible to modification by other processors.

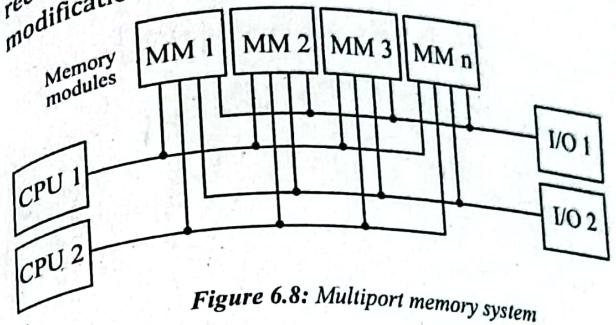

Figure 6.8: Multiport memory system

3. **Central Control Unit**

It manages the transfer of separate data streams back and forth between independent modules like CPU, memory and I/O. The controller can buffer requests and perform arbitration and timing functions. It can also pass status and control messages between CPUs. All the co-ordination is concentrated in the central control unit un-disturbing the modules. It is more flexible and complex as well.

6.6 Digital Signal Processor (DSP)

A *digital signal processor* (DSP) is a specialized microprocessor with an architecture optimized for the fast operational needs of digital signal processing. Essentially, a DSP is a special type of microprocessor that processes data in real time. Its applications focus on the processing of digital data that represents analog signals. A DSP, like a microprocessor, has a central processing unit (CPU) and memory units in addition to many interfacing functions. Every time, we use our cellular telephone, we are using a DSP, and this is only one example of its many applications.

The digital signal processor (DSP) is the heart of a digital signal processing system. It takes its input from an ADC and produces an output that goes to a DAC, as shown in Figure 1.9.

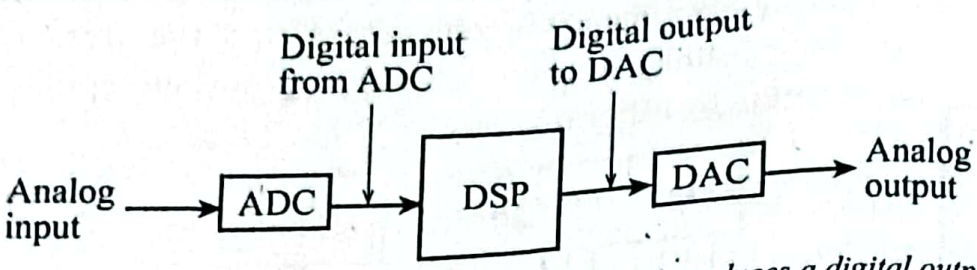

Figure 6.9: The DSP has a digital input and produces a digital output.

DSPs are typically programmed in either assembly language or in C. Because programs written in assembly language can usually execute faster and because speed is critical in most DSP applications, assembly language is used much more in DSPs than in general-purpose microprocessors. Also, DSP programs are usually much shorter than traditional microprocessor programs because of their very specialized applications where much redundancy is used. In general, the instruction sets for DSPs tend to be smaller than for microprocessors.

DSP applications:

The DSP, unlike the general-purpose microprocessor, must typically process data in real time; that is, as it happens. Many applications in which DSPs are used cannot tolerate any noticeable delays, requiring the DSP to be extremely fast. In addition to cell phones, digital signal processors (DSPs) are used in multimedia computers, video recorders, CD players, hard disk drives, digital radio modems, and other applications to improve the signal quality. DSPs are also used in television applications. For example, television converters use DSP to provide compatibility with various television standards.

An important application of DSPs is in signal compression and decompression. In CD systems, for example, the music on the CD is in a compressed form so that it doesn't use as much storage space. It must be decompressed in order to be reproduced. Also signal compression is used in cell phones to allow a greater number of calls to be handled simultaneously in a local cell.