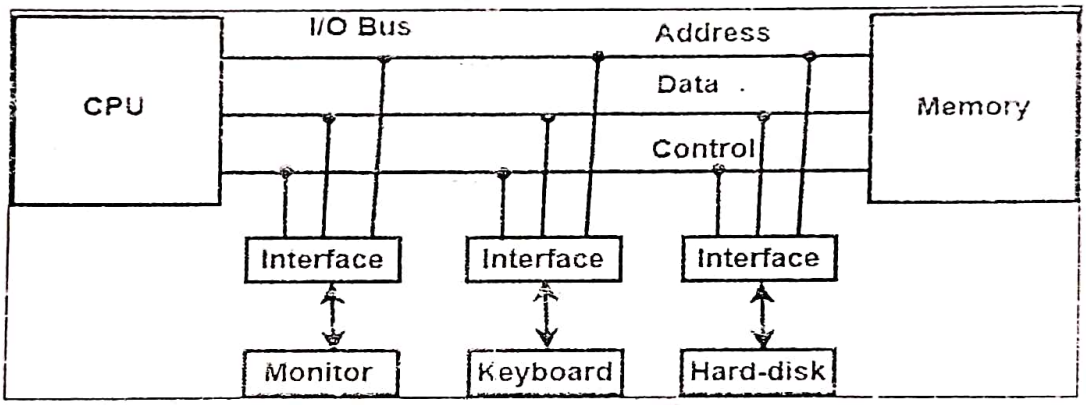
Input-output organization: I/O interface. I/O Bus and interface modules. I/O versus Memory Bus. Asynchronous data transfer. Strobe control. Handshaking. Asynchronous serial transfer. Modes of Transfer: Programmed I/O. Interrupt driven I/O. Priority interrupt: Daisy chaining. Parallel Priority interrupt. Direct memory Access. DMA controller and transfer. Input output Processor. CPU-IOP communication. Serial communication.

I/O INTERFACE

Input-Output Interface is used as an method which helps in transferring of information between the internal storage devices i.e. memory and the external peripheral device . A peripheral device is that which provide input and output for the computer, it is also called Input-Output devices. For Example: A keyboard and mouse provide Input to the computer are called input devices while a monitor and printer that provide output to the computer are called output devices. Just like the external hard-drives, there is also availability of some peripheral devices which are able to provide both input and output.



In micro-computer base system, the only purpose of peripheral devices is just to provide special **communication links** for the interfacing them with the CPU. To resolve the differences between peripheral devices and CPU, there is a special need for communication links.

The major differences are as follows:

1. The nature of peripheral devices is electromagnetic and electro-mechanical. The nature of the CPU is electronic. There is a lot of difference in the mode of operation of both peripheral devices and CPU.
2. There is also a synchronization mechanism because the data transfer rate of peripheral devices are slow than CPU.
3. In peripheral devices, data code and formats are differ from the format in the CPU and memory.

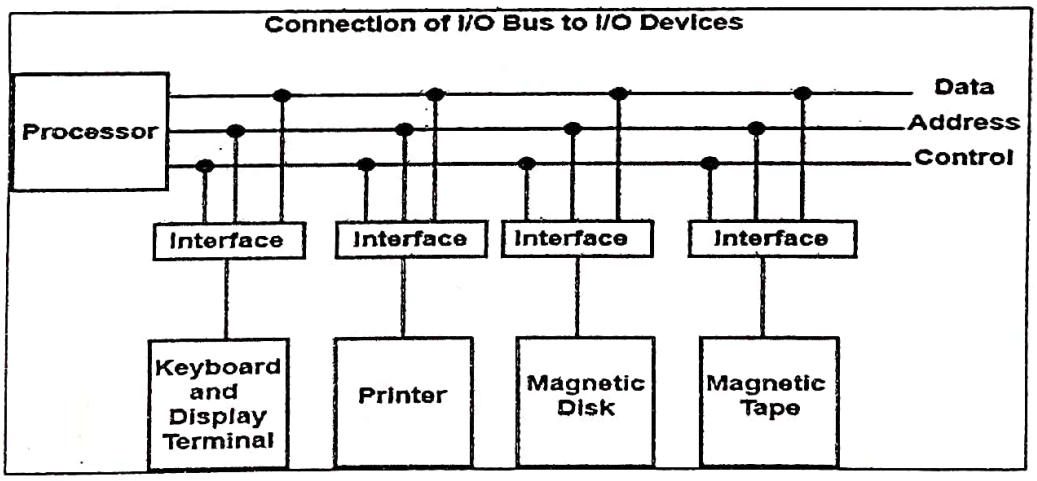
4. The operating mode of peripheral devices are different and each may be controlled so as not to disturb the operation of other peripheral devices connected to CPU.
5. There is a special need of the additional hardware to resolve the differences between CPU and peripheral devices to supervise and synchronize all input and output devices.

Functions of Input-Output Interface:

1. It is used to synchronize the operating speed of CPU with respect to input-output devices.
2. It selects the input-output device which is appropriate for the interpretation of the input-output device.
3. It is capable of providing signals like control and timing signals.
4. In this data buffering can be possible through data bus.
5. There are various error detectors.
6. It converts serial data into parallel data and vice-versa.
7. It also convert digital data into analog signal and vice-versa.

I/O BUS AND INTERFACE MODULES

The I/O bus is the route used for peripheral devices to interact with the computer processor. A typical connection of the I/O bus to I/O devices is shown in the figure.



The I/O bus includes data lines, address lines, and control lines. In any general-purpose computer, the magnetic disk, printer, and keyboard, and display terminal are commonly employed. Each peripheral unit has an interface unit associated with it. Each interface decodes the control and address received from the I/O bus.

It can describe the address and control received from the peripheral and supports signals for the peripheral controller. It also conducts the transfer of information between peripheral and processor and also integrates the data flow.

The I/O bus is linked to all peripheral interfaces from the processor. The processor locates a device address on the address line to interact with a specific device. Each interface contains an address decoder attached to the I/O bus that monitors the address lines.

When the address is recognized by the interface, it activates the direction between the bus lines and the device that it controls. The interface disables the peripherals whose address does not equivalent to the address in the bus.

An interface receives any of the following four commands –

- **Control** – A command control is given to activate the peripheral and to inform its next task. This control command depends on the peripheral, and each peripheral receives its sequence of control commands, depending on its mode of operation.
- **Status** – A status command can test multiple test conditions in the interface and the peripheral.
- **Data Output** – A data output command creates the interface counter to the command by sending data from the bus to one of its registers.
- **Data Input** – The data input command is opposite to the data output command. In data input, the interface gets an element of data from the peripheral and places it in its buffer register.

I/O VERSUS MEMORY BUS

Memory mapped I/O and Isolated I/O

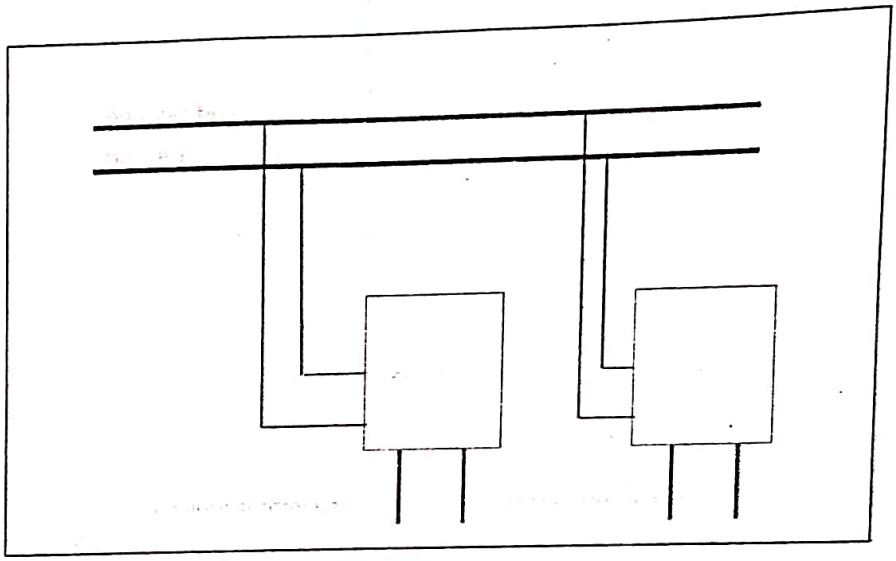
As a CPU needs to communicate with the various memory and input-output devices (I/O) the data between the processor and these devices flow with the help of the system bus. There are three ways in which system bus can be allotted to them :

1. Separate set of address, control and data bus to I/O and memory.
2. Have common bus (data and address) for I/O and memory but separate control lines.
3. Have common bus (data, address, and control) for I/O and memory.

In first case it is simple because both have different set of address space and instruction but require more buses.

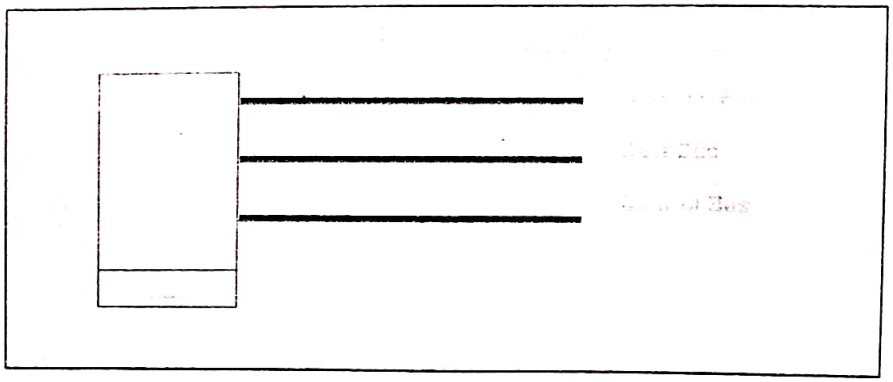
Isolated I/O –

In Isolated I/O in common bus (data and address) is used for I/O and memory but separate read and write control lines for I/O. So when CPU decode instruction then if data is for I/O then it places the address on the address line and set I/O read or write control line on due to which data transfer occurs between CPU and I/O. As the address space of memory and I/O is isolated and the name is so. The address for I/O here is called ports. Here we have different read-write instruction for both I/O and memory.



Memory Mapped I/O –

In this case every bus in common due to which the same set of instructions work for memory and I/O. Hence we manipulate I/O same as memory and both have same address space, due to which addressing capability of memory become less because some part is occupied by the I/O.



Differences between memory mapped I/O and isolated I/O –

Isolated I/O	Memory Mapped I/O
Memory and I/O have separate address space	Both have same address space
All address can be used by the memory	Due to addition of I/O addressable memory become less for memory
Separate instruction control read and write operation in I/O and Memory	Same instructions can control both I/O and Memory
In this I/O address are called ports.	Normal memory address are for both
More efficient due to separate buses	Lesser efficient
Larger in size due to more buses	Smaller in size
It is complex due to separate logic is used to control both.	Simpler logic is used as I/O is also treated as memory only.

ASYNCHRONOUS DATA TRANSFER: STROBE CONTROL, HANDSHAKING

The internal operations in an individual unit of a digital system are synchronized using clock pulse. It means clock pulse is given to all registers within a unit. And all data transfer among internal registers occurs simultaneously during the occurrence of the clock pulse. Now, suppose any two units of a digital system are designed independently, such as CPU and I/O interface.

If the registers in the I/O interface share a common clock with CPU registers, then transfer between the two units is said to be synchronous. But in most cases, the internal timing in each unit is independent of each other, so each uses its private clock for its internal registers. In this case, the two units are said to be asynchronous to each other, and if data transfer occurs between them, this data transfer is called **Asynchronous Data Transfer.**

But, the Asynchronous Data Transfer between two independent units requires that control signals be transmitted between the communicating units so that the time can be indicated at which they send data. These two methods can achieve this asynchronous way of data transfer:

- **Strobe control:** A strobe pulse is supplied by one unit to indicate to the other unit when the transfer has to occur.
- **Handshaking:** This method is commonly used to accompany each data item being transferred with a control signal that indicates data in the bus. The unit receiving the data item responds with another signal to acknowledge receipt of the data.

The strobe pulse and handshaking method of asynchronous data transfer is not restricted to I/O transfer. They are used extensively on numerous occasions requiring the transfer of data between two independent units. So, here we consider the transmitting unit as a source and receiving unit as a destination.

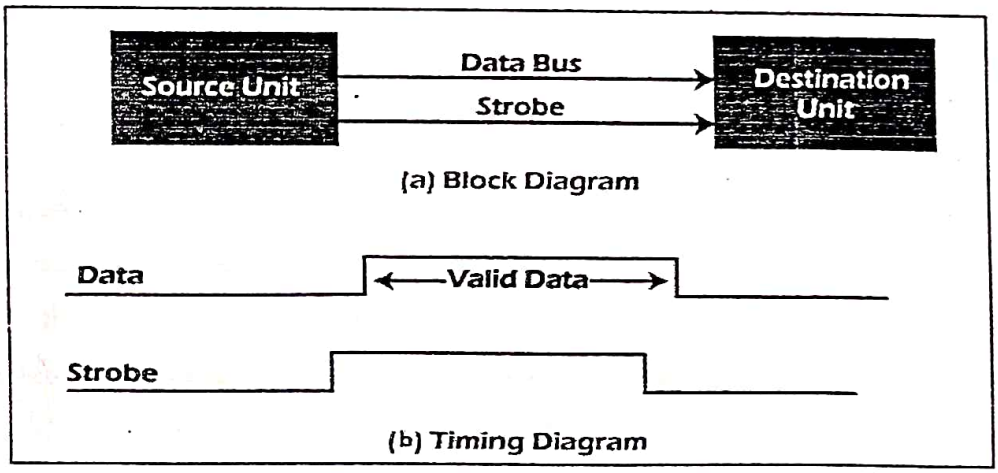
Asynchronous Data Transfer Methods

The asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate when they send the data. Thus, the two methods can achieve the asynchronous way of data transfer.

1. Strobe Control Method

The Strobe Control method of asynchronous data transfer employs a single control line to time each transfer. This control line is also known as a strobe, and it may be achieved either by source or destination, depending on which initiate the transfer.

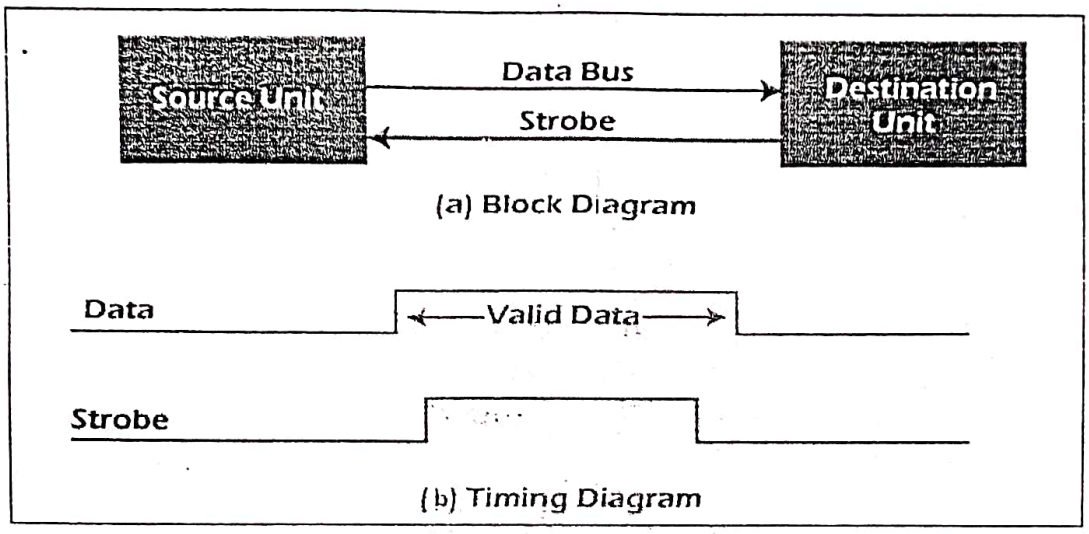
- **Source initiated strobe:** In the below block diagram, you can see that strobe is initiated by source, and as shown in the timing diagram, the source unit first places the data on the data bus.



After a brief delay to ensure that the data resolve to a stable value, the source activates a strobe pulse. The information on the data bus and strobe control signal remains in the active state for a sufficient time to allow the destination unit to receive the data. The destination unit uses a falling

edge of strobe control to transfer the contents of a data bus to one of its internal registers. The source removes the data from the data bus after it disables its strobe pulse. Thus, new valid data will be available only after the strobe is enabled again. In this case, the strobe may be a memory-write control signal from the CPU to a memory unit. The CPU places the word on the data bus and informs the memory unit, which is the destination.

- **Destination initiated strobe:** In the below block diagram, you see that the strobe initiated by destination, and in the timing diagram, the destination unit first activates the strobe pulse, informing the source to provide the data.



The source unit responds by placing the requested binary information on the data bus. The data must be valid and remain on the bus long enough for the destination unit to accept it. The falling edge of the strobe pulse can use again to trigger a destination register. The destination unit then disables the strobe. Finally, and source removes the data from the data bus after a determined time interval. In this case, the strobe may be a memory read control from the CPU to a memory unit. The CPU initiates the read operation to inform the memory, which is a source unit, to place the selected word into the data bus.

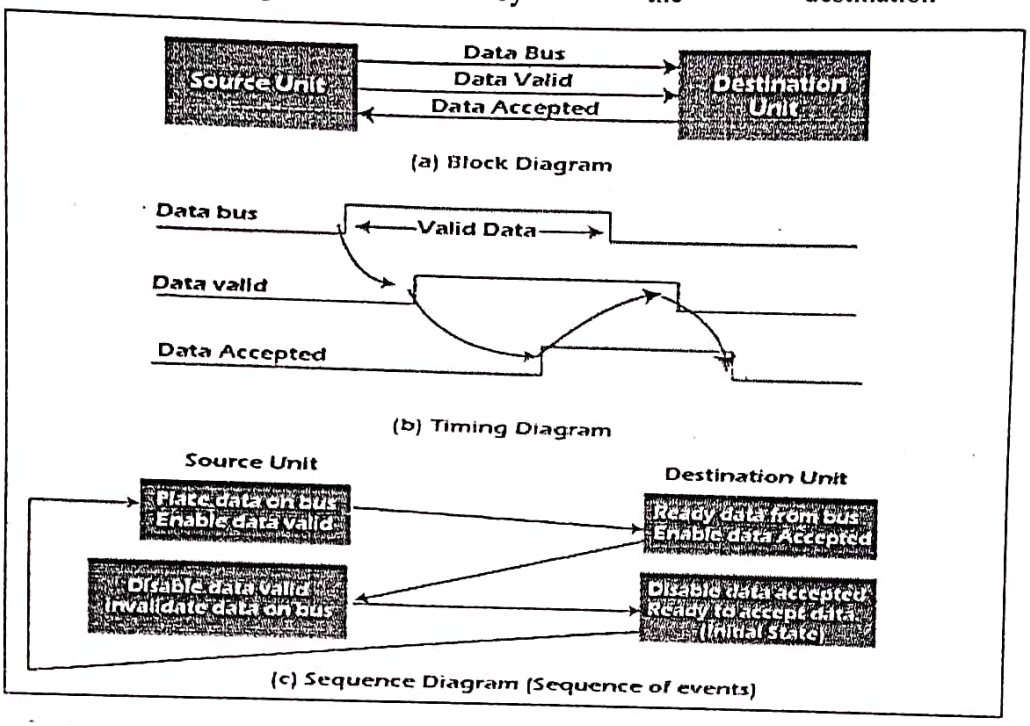
2. Handshaking Method

The strobe method has the disadvantage that the source unit that initiates the transfer has no way of knowing whether the destination has received the data that was placed in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has placed data on the bus. So this problem is solved by the handshaking method. The handshaking method introduces a second control signal line that replays the unit that initiates the transfer.

In this method, one control line is in the same direction as the data flow in the bus from the source to the destination. The source unit uses it to inform the destination unit whether there are

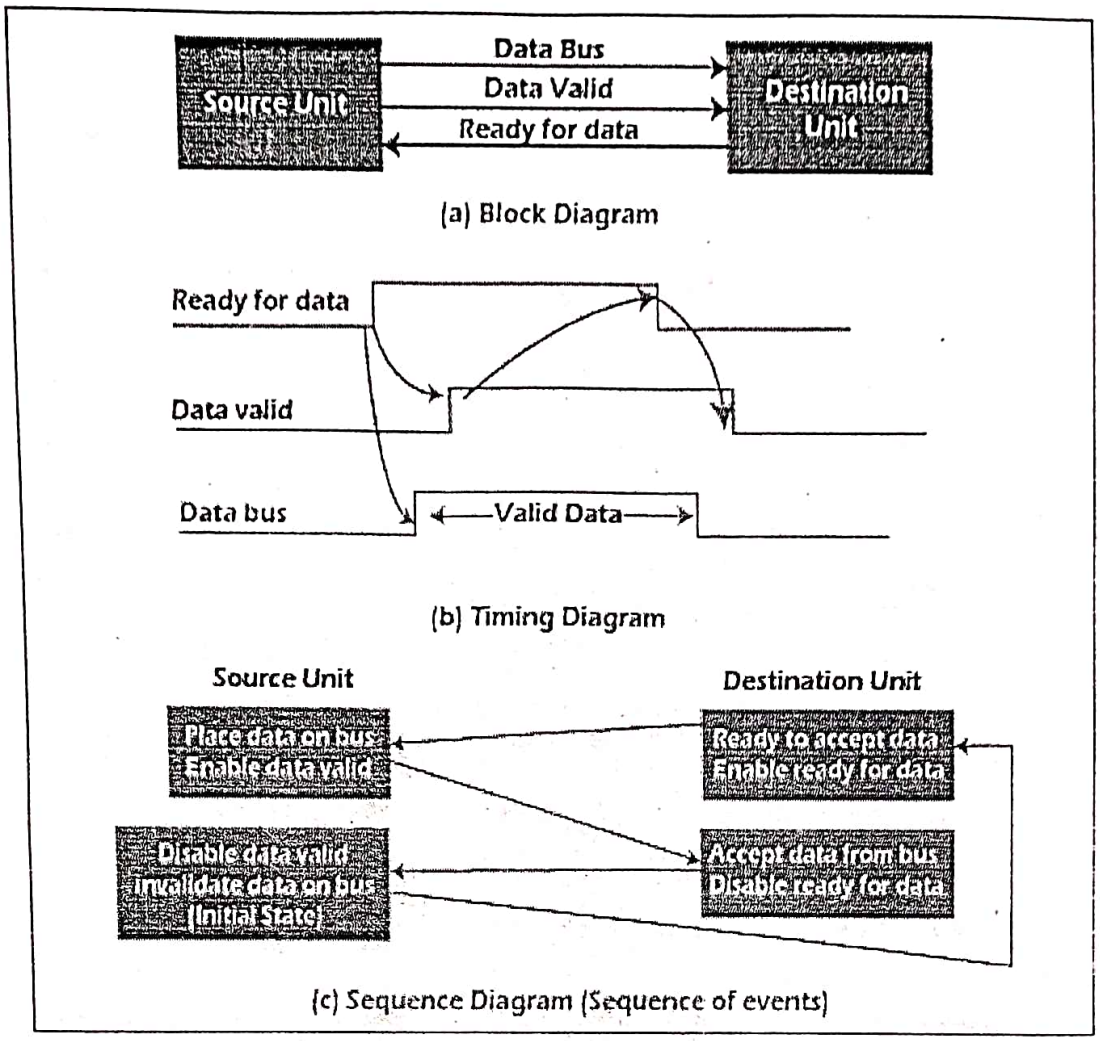
valid data in the bus. The other control line is in the other direction from the destination to the source. This is because the destination unit uses it to inform the source whether it can accept data. And in it also, the sequence of control depends on the unit that initiates the transfer. So it means the sequence of control depends on whether the transfer is initiated by source and destination.

- **Source initiated handshaking:** In the below block diagram, you can see that two handshaking lines are "**data valid**", which is generated by the source unit, and "**data accepted**", generated by the destination unit.



The timing diagram shows the timing relationship of the exchange of signals between the two units. The source initiates a transfer by placing data on the bus and enabling its data valid signal. The destination unit then activates the data accepted signal after it accepts the data from the bus. The source unit then disables its valid data signal, which invalidates the data on the bus. After this, the destination unit disables its data accepted signal, and the system goes into its initial state. The source unit does not send the next data item until after the destination unit shows readiness to accept new data by disabling the data accepted signal. This sequence of events described in its sequence diagram, which shows the above sequence in which the system is present at any given time.

- **Destination initiated handshaking:** In the below block diagram, you see that the two handshaking lines are "**data valid**", generated by the source unit, and "**ready for data**" generated by the destination unit. Note that the name of signal data accepted generated by the destination unit has been changed to ready for data to reflect its new meaning.



The destination transfer is initiated, so the source unit does not place data on the data bus until it receives a ready data signal from the destination unit. After that, the handshaking process is the same as that of the source initiated. The sequence of events is shown in its sequence diagram, and the timing relationship between signals is shown in its timing diagram. Therefore, the sequence of events in both cases would be identical.

Advantages of Asynchronous Data Transfer

Asynchronous Data Transfer in computer organization has the following advantages. such as:

- It is more flexible, and devices can exchange information at their own pace. In addition, individual data characters can complete themselves so that even if one packet is corrupted, its predecessors and successors will not be affected.
- It does not require complex processes by the receiving device. Furthermore, it means that inconsistency in data transfer does not result in a big crisis since the device can keep up with the data stream. It also makes asynchronous transfers suitable for applications where character data is generated irregularly.

Disadvantages of Asynchronous Data Transfer

There are also some disadvantages of using asynchronous data for transfer in computer organization, such as:

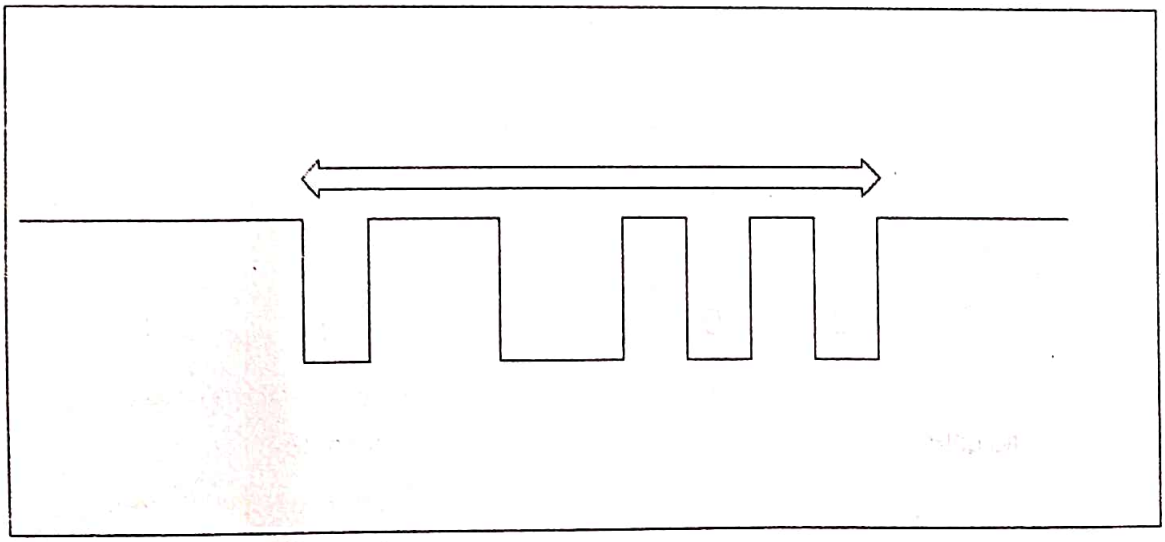
- The success of these transmissions depends on the start bits and their recognition. Unfortunately, this can be easily susceptible to line interference, causing these bits to be corrupted or distorted.
- A large portion of the transmitted data is used to control and identify header bits and thus carries no helpful information related to the transmitted data. This invariably means that more data packets need to be sent.

ASYNCHRONOUS SERIAL TRANSFER

In most computer asynchronous mode of data transfer is used in which two component have a different clock. Data transfer can occur between data in two ways serial and parallel. In case of parallel multiple lines are used to send a single bit whereas in serial transfer each bit is send one at a time. To tell other devices when the character/data will be given a concept of start and end bit is used. A start bit is denoted by 0 and stop bit is detected when line return to 1-state at least one time. here 1-state means that there is not data transfer is occurring.

When a character is not being sent then line is kept in state 1. Start of character is detected when a 0 is sent. The character bit always come after 0 bit. After last bit is sent the state of line to become 1.

The diagram below shows this concept:



Here earlier state of line was 1 when a character has to be send a 0 is send and character bit are transferred.

Difference between serial and parallel transfer –

Serial	Parallel
Require single line to send data	Require multiple line
Less error and simple model	Error prone and complex working
Economical	Expensive
Slower data transfer	Faster data transfer
Used for long distance	used for short distance
Example: Computer to Computer	Example: Computer to Printer

MODES OF TRANSFER: PROGRAMMED I/O, INTERRUPT DRIVEN I/O

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

1. **Programmed I/O.**
2. **Interrupt- initiated I/O.**
3. **Direct memory access(DMA).**

1. Programmed I/O: It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.

Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the

program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.

2. Interrupt- initiated I/O: Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

Note: Both the methods programmed I/O and Interrupt-driven I/O require the active intervention of the processor to transfer data between memory and the I/O module, and any data transfer must transverse a path through the processor. Thus both these forms of I/O suffer from two inherent drawbacks.

- The I/O transfer rate is limited by the speed with which the processor can test and service a device.
- The processor is tied up in managing an I/O transfer; a number of instructions must be executed
  for each I/O transfer.

3. Direct Memory Access: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

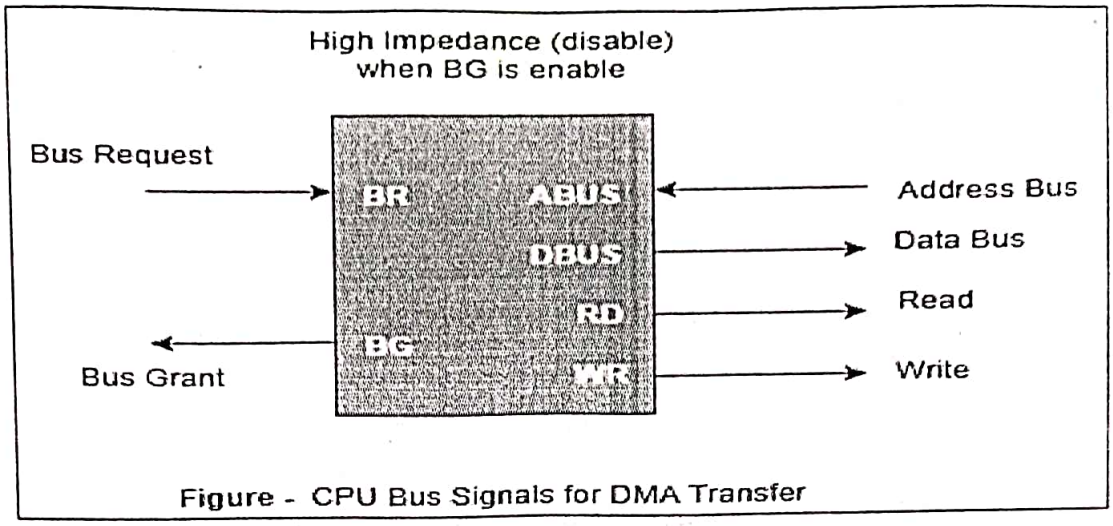


Figure - CPU Bus Signals for DMA Transfer

Bus Request: It is used by the DMA controller to request the CPU to relinquish the control of the buses.

Bus Grant: It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

PRIORITY INTERRUPT: DAISY CHAINING, PARALLEL PRIORITY INTERRUPT

A priority interrupt is a system which decides the priority at which various devices, which generates the interrupt signal at the same time, will be serviced by the CPU. The system has authority to decide which conditions are allowed to interrupt the CPU, while some other interrupt is being serviced. Generally, devices with high speed transfer such as magnetic disks are given high priority and slow devices such as keyboards are given low priority.

When two or more devices interrupt the computer simultaneously, the computer services the device with the higher priority first.

Types of Interrupts:

1. **Hardware Interrupts** 2. **Software Interrupts**

1. Hardware Interrupts:

When the signal for the processor is from an external device or hardware then this interrupts is known as hardware interrupt.

Let us consider an example: when we press any key on our keyboard to do some action, then this pressing of the key will generate an interrupt signal for the processor to perform certain action. Such an interrupt can be of two types:

- **Maskable Interrupt**

The hardware interrupts which can be delayed when a much high priority interrupt has occurred at the same time.

- **Non Maskable Interrupt**

The hardware interrupts which cannot be delayed and should be processed by the processor immediately.

2. Software Interrupts

The interrupt that is caused by any internal system of the computer system is known as a software interrupt. It can also be of two types:

- **Normal Interrupt**

The interrupts that are caused by software instructions are called normal software interrupts.

- **Exception**

Unplanned interrupts which are produced during the execution of some program are called exceptions, such as division by zero.

Daisy chaining

The daisy-chaining method of establishing priority consists of a serial connection of all devices that request an interrupt. The device with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the chain. This method of connection between three devices and the CPU is shown in Fig. 12. The interrupt request line is common to all devices and forms a wired logic connection. If any device has its interrupt signal in the low-level state, the interrupt line goes to the low-level state and enables the interrupt input in the CPU. When no interrupts are pending, the interrupt line stays in the high-level state and no interrupts are recognized by the CPU. This is equivalent to a negative logic OR operation. The CPU responds to an interrupt request by enabling the interrupt acknowledge line. This signal is received by device 1 at its PI (priority in) input. The acknowledge signal passes on to the next device through the PO (priority out) output only if device 1 is not requesting an interrupt. If device 1 has a pending interrupt, it blocks the acknowledge signal from the next device by placing a 0 in the PO output. It then proceeds to insert its own interrupt vector address (VAD) into the data bus for the CPU to use during the interrupt cycle. A device with a 0 in its PI input generates a 0 in its PO output to inform the next-lower-priority device that the acknowledge signal has been blocked. A device that is requesting an interrupt and has a 1 in its PI input will intercept the acknowledge signal by placing a 0 in its PO output. If the device does not have pending interrupts, it transmits the acknowledge signal to the next device by placing a 1 in its PO output.

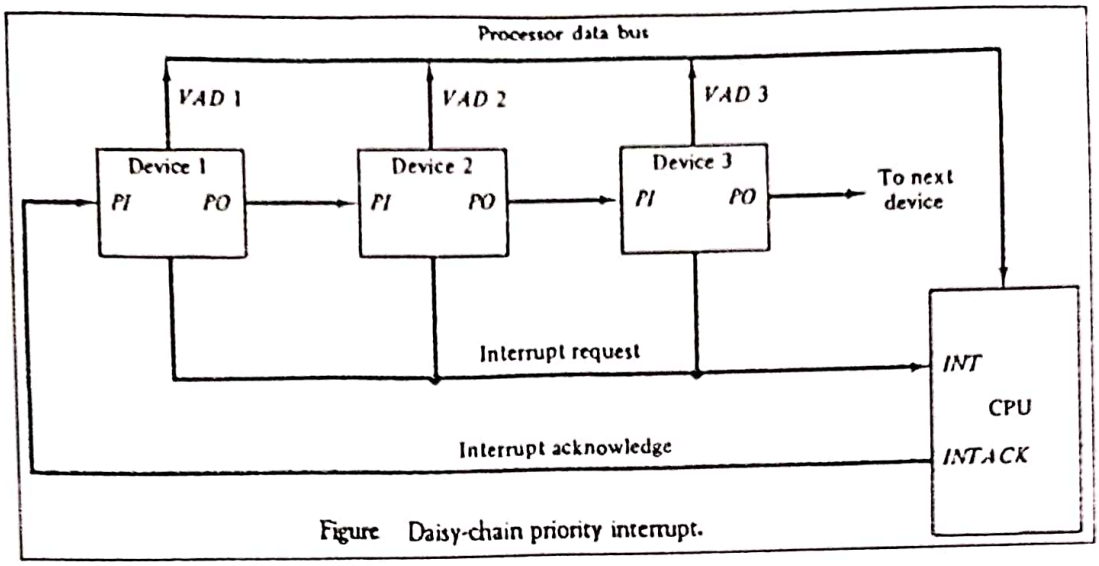


Figure Daisy-chain priority interrupt.

Parallel priority interrupt

The parallel priority interrupt method uses a register whose bits are set separately by the interrupt signal from each device. Priority is established according to the position of the bits in the register. In addition to the interrupt register, the circuit may include a mask register whose purpose is to control the status of each interrupt request.

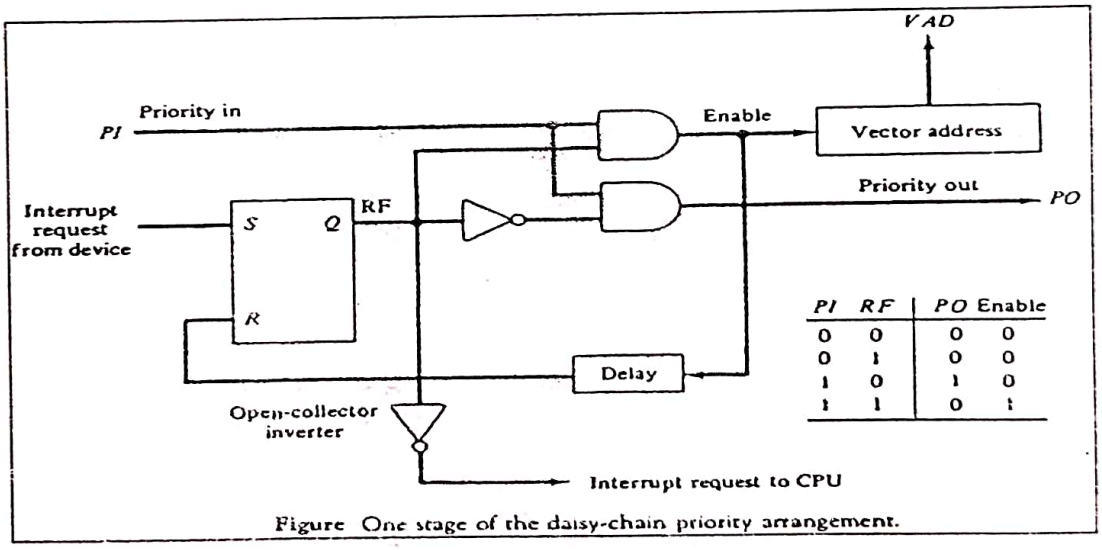


Figure One stage of the daisy-chain priority arrangement.

The mask register can be programmed to disable lower-priority interrupts while a higher-priority device is being serviced. It can also provide a facility that allows a high-priority device to interrupt the CPU while a lower-priority device is being serviced. It consists of an interrupt register whose individual bits are set by external conditions and cleared by program instructions. The magnetic disk, being a high-speed device, is given the highest priority. The printer has the next priority, followed by a character reader and a keyboard. The mask register has the same number of bits as the interrupt register. By means of program instructions, it is possible to set or

reset any bit in the mask register. Each interrupt bit and its corresponding mask bit are applied to an AND gate to produce the four inputs to a priority encoder. In this way an interrupt is recognized only if its corresponding mask bit is set to 1 by the program. The priority encoder generates two bits of the vector address, which is transferred to the CPU. Another output from the encoder sets an interrupt status flip-flop IST when an interrupt that is not masked occurs. The interrupt enable flip-flop IEN can be set or cleared by the program to provide an overall control over the interrupt system. The outputs of IST ANDed with IEN provide a common interrupt signal for the CPU. The interrupt acknowledge INTACK signal from the CPU enables the bus buffers in the output register and a vector address VAD is placed into the data bus.

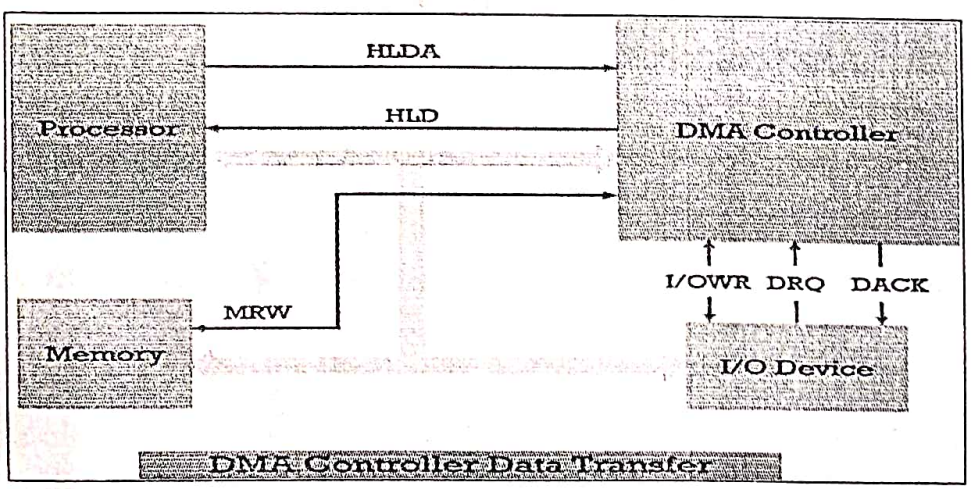
DIRECT MEMORY ACCESS, DMA CONTROLLER AND TRANSFER

DMA stands for Direct Memory Access. It is designed by Intel to transfer data at the fastest rate. It allows the device to transfer the data directly to/from memory without any interference of the CPU. Using a DMA controller, the device requests the CPU to hold its data, address and control bus, so the device is free to transfer data directly to/from the memory. The DMA data transfer is initiated only after receiving HLDA signal from the CPU.

Direct Memory Access (DMA) transfers the block of data between the memory and peripheral devices of the system, **without the participation** of the **processor.** The unit that controls the activity of accessing memory directly is called a **DMA controller.**

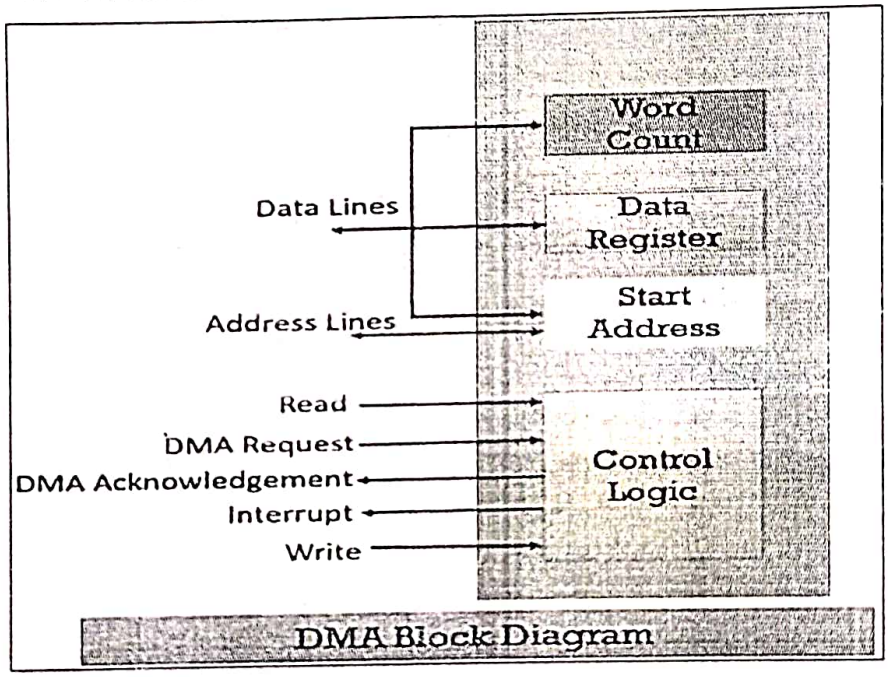
DMA Controller

DMA controller is a **hardware unit** that allows I/O devices to access memory directly without the participation of the processor. Here, we will discuss the working of the DMA controller. Below we have the diagram of DMA controller that explains its working:



1. Whenever an I/O device wants to transfer the data to or from memory, it sends the DMA request (DRQ) to the DMA controller. DMA controller accepts this DRQ and asks the CPU to hold for a few clock cycles by sending it the Hold request (HLD).
2. CPU receives the Hold request (HLD) from DMA controller and relinquishes the bus and sends the Hold acknowledgement (HLDA) to DMA controller.
3. After receiving the Hold acknowledgement (HLDA), DMA controller acknowledges I/O device (DACK) that the data transfer can be performed and DMA controller takes the charge of the system bus and transfers the data to or from memory.
4. When the data transfer is accomplished, the DMA raise an interrupt to let know the processor that the task of data transfer is finished and the processor can take control over the bus again and start processing where it has left.

Now the DMA controller can be a separate unit that is shared by various I/O devices, or it can also be a part of the I/O device interface.



Whenever a processor is requested to read or write a block of data, i.e. transfer a block of data, it instructs the DMA controller by sending the following information.

1. The first information is whether the data has to be read from memory or the data has to be written to the memory. It passes this information via read or write control lines that is between the processor and DMA controllers control logic unit.
2. The processor also provides the starting address of/ for the data block in the memory. from where the data block in memory has to be read or where the data block has to be written in memory. DMA controller stores this in its address register. It is also called the starting address register.

3. The processor also sends the word count, i.e. how many words are to be read or written. It stores this information in the data count or the word count register.
4. The most important is the address of I/O device that wants to read or write data. This information is stored in the data register.

Direct Memory Access Advantages and Disadvantages

Advantages:

1. Transferring the data without the involvement of the processor will **speed up** the read-write task.
2. DMA **reduces the clock cycle** requires to read or write a block of data.
3. Implementing DMA also **reduces the overhead** of the processor.

Disadvantages

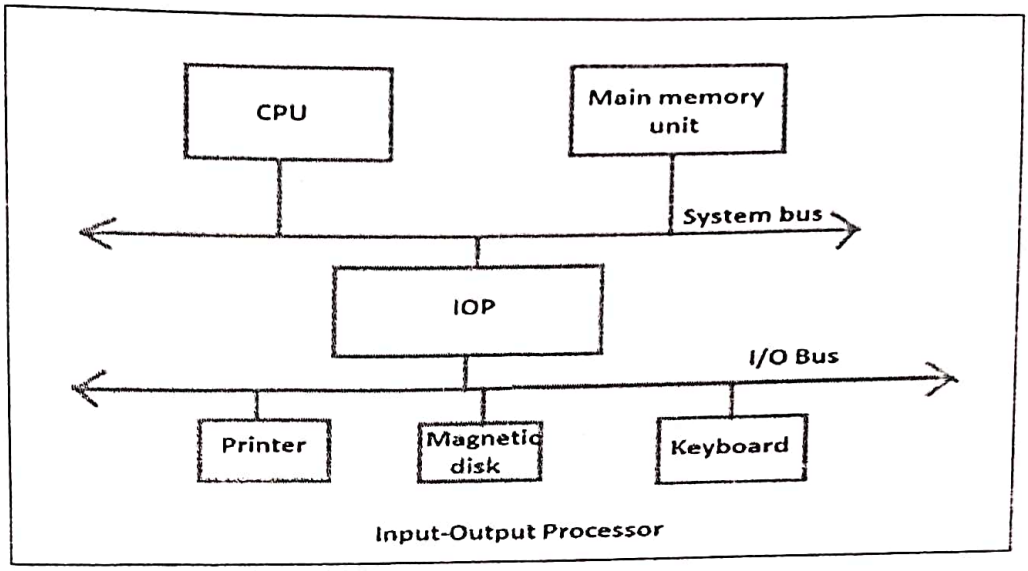
1. As it is a hardware unit, it would **cost** to implement a DMA controller in the system.
2. Cache **coherence** problem can occur while using DMA controller.

INPUT OUTPUT PROCESSOR, CPU-IOP COMMUNICATION, SERIAL COMMUNICATION

The **DMA mode** of data transfer reduces CPU's overhead in handling I/O operations. It also allows parallelism in CPU and I/O operations. Such parallelism is necessary to avoid wastage of valuable CPU time while handling I/O devices whose speeds are much slower as compared to CPU. The concept of DMA operation can be extended to relieve the CPU further from getting involved with the execution of I/O operations. This gives rises to the development of special purpose processor called **Input-Output Processor (IOP) or IO channel.**

The Input Output Processor (IOP) is just like a CPU that handles the details of I/O operations. It is more equipped with facilities than those are available in typical DMA controller. The IOP can fetch and execute its own instructions that are specifically designed to characterize I/O transfers. In addition to the I/O – related tasks, it can perform other processing tasks like arithmetic, logic, branching and code translation. The main memory unit takes the pivotal role. It communicates with processor by the means of DMA.

The block diagram –



The Input Output Processor is a specialized processor which loads and stores data into memory along with the execution of I/O instructions. It acts as an interface between system and devices. It involves a sequence of events to executing I/O operations and then store the results into the memory.

Advantages –

- The I/O devices can directly access the main memory without the intervention by the processor in I/O processor based systems.
- It is used to address the problems that are arises in Direct memory access method.

CPU-IOP Communication

There is a communication channel between IOP and CPU to perform task which come under computer architecture. This channel explains the commands executed by IOP and CPU while performing dome programs. The CPU do not executes the instructions but it assigns the task of initiating operations, the instructions are executed by IOP. I/O transfer is instructed by CPU. The IOP asks for CPU through interrupt.

This channel starts by CPU, by giving "**test IOP path**" instruction to IOP and then the communication begins as shown in diagram:

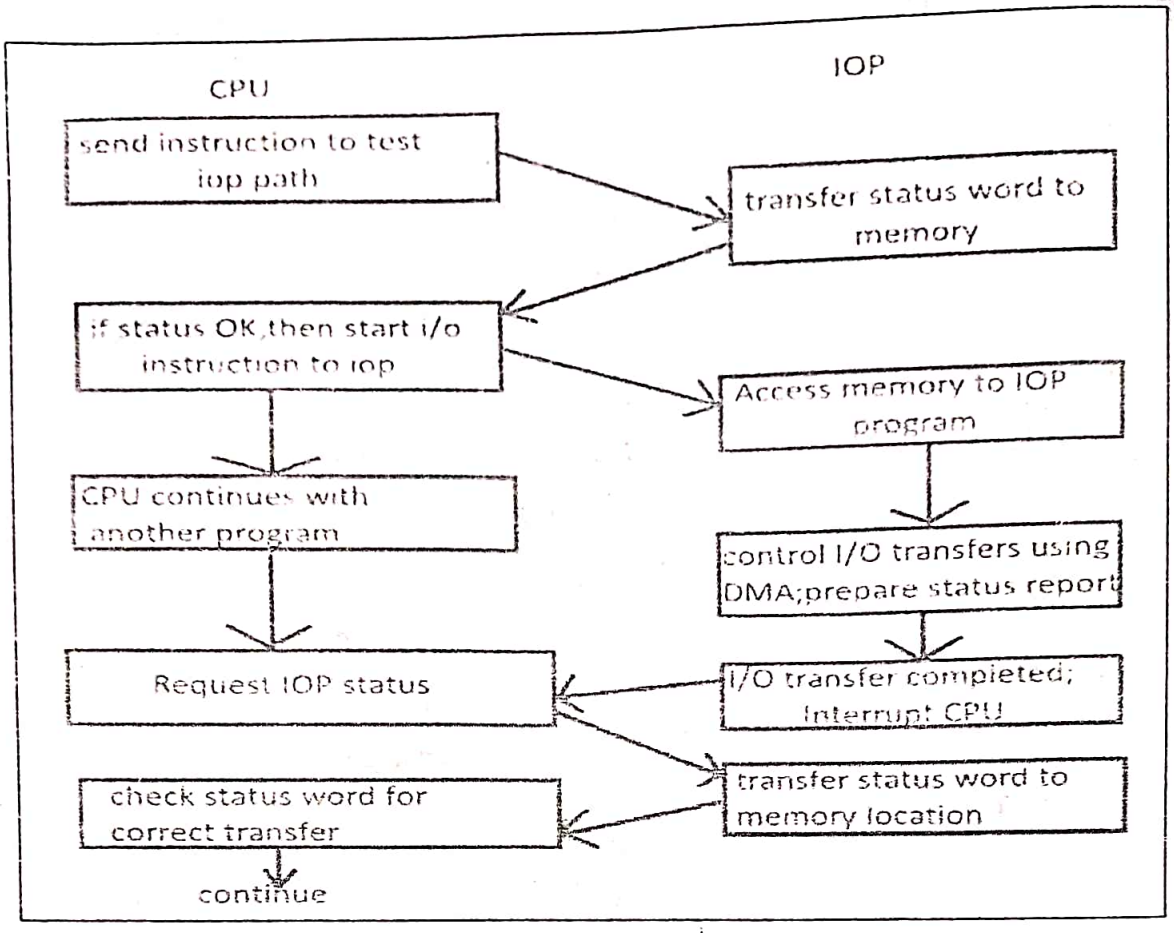


Figure – Communication channel between IOP and CPU

Whenever CPU gets interrupt from IOP to access memory, it sends test path instruction to IOP. IOP executes and check for status, if the status given to CPU is OK, then CPU gives start instruction to IOP and gives it some control and get back to some another (or same) program, after that IOP is able to access memory for its program.

Now IOP start controlling I/O transfer using DMA and create another status report as well. AS soon as this I/O transfer completes IOP once again send interrupt to CPU, CPU again request for status of IOP, IOP check status word from memory location and gives it to CPU. Now CPU check the correctness of status and continues with the same process.