



**Subject :
Computer Organization & Architecture**

**Designed by :
Mr. Shashank Sharma**

**Designation :
Assistant Professor**



UNIT 3

I/O Organization:

A decorative border surrounds the central text. It features a black line forming a frame with rounded corners. Various geometric shapes are placed along this border: a red circle at the top-left, a black circle at the top-right, a red diamond, a pink circle, and a black diamond on the left side, and a pink diamond, a red circle, and a black diamond on the bottom-right. Curved arrows are positioned at the top-left, top-right, and bottom-left corners, pointing inwards towards the center.

Syllabus

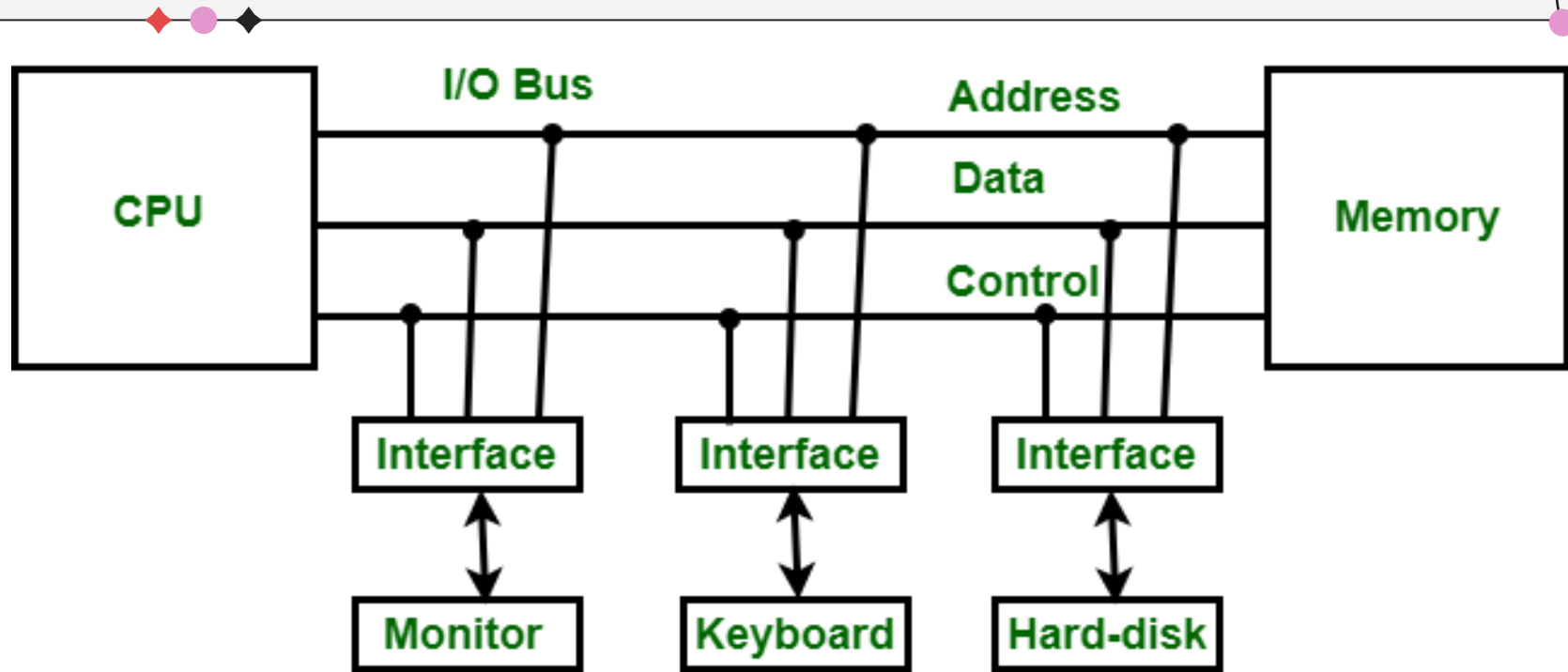
- **I/O Interface** –PCI Bus, SCSI Bus, USB,
- **Data Transfer** : Serial, Parallel, Synchronous, Asynchronous, Modes of Data Transfer,
 - **Direct Memory Access(DMA)**
 - **I/O Processor.**

I/O Interface

I/O interface used as a method which help in transferring of information between the internal storage device i.e. memory and external peripheral device.

A **Peripheral** device is that which provide input and output for the computer, it is also called Input-Output devices.

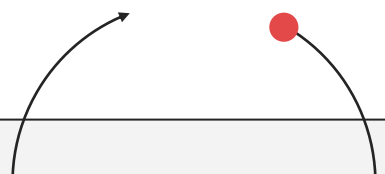
For Example: A keyboard and mouse provide Input to the computer are called **input devices** while a monitor and printer that provide output to the computer are called **output devices**. Just like the external hard-drives, there is also availability of some peripheral devices which are able to provide both input and output.



Input-Output Interface



➤ Functions of Input-Output Interface:

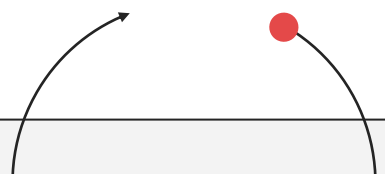
- 1.It is used to synchronize the operating speed of CPU with respect to input-output devices.
 - 2.It selects the input-output device which is appropriate for the interpretation of the input-output signal.
 - 3.It is capable of providing signals like control and timing signals.
 - 4.In this data buffering 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.
- 



PCI (Peripheral Component Interconnect) :

The PCI bus is a rapid hardware interface that facilitates the attachment of diverse peripheral components to a computer's motherboard.

Launched in 1992, the PCI bus emerged as the standard method for connecting expansion cards, including graphics cards, network cards, and sound cards. This interface offers plug-and-play capabilities and allows for concurrent data transmission among several devices.

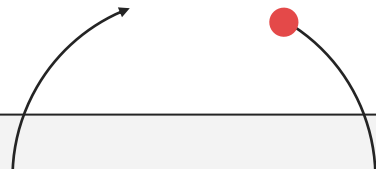




Pros:

- High-speed data transfer.
- Compatibility with a wide range of hardware devices.
- Easy to install and configure.

Cons:

- Limited data transfer rates compared to newer technologies like PCIe.
 - Declining support in modern computer systems.
- 



Features:

High bandwidth: PCI offers high-speed data transfer rates, making it suitable for demanding applications.

Plug and Play: PCI supports Plug and Play functionality, allowing devices to be automatically detected and configured by the operating system.

Bus mastering: PCI allows devices to take control of the bus for direct memory access (DMA) transfers, reducing CPU overhead.



Hot swapping: Some PCI implementations support hotswapping, allowing devices to be added or removed from the system without shutting down.



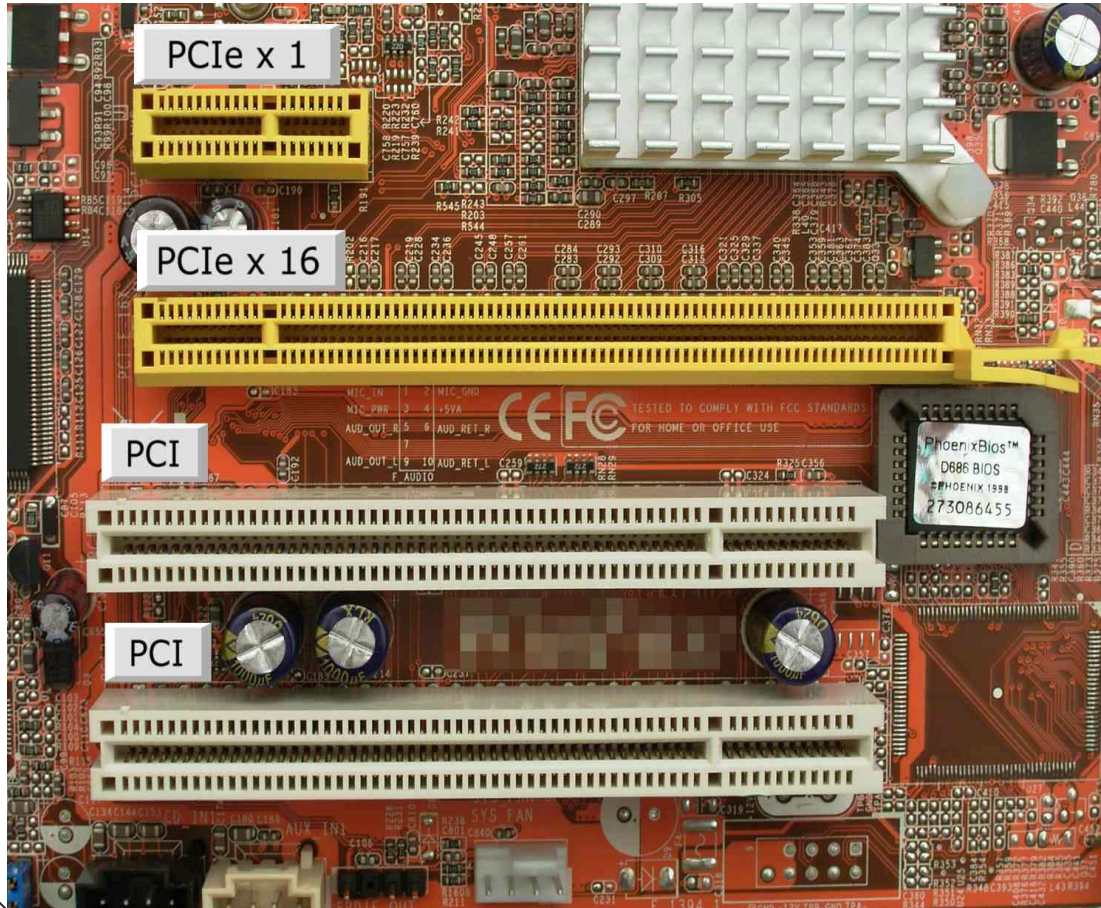
Variants

PCI: The original PCI standard, offering data transfer rates of up to 133 MB/s.

PCI-X: PCI Extended, an extension of the original PCI standard, offering higher speeds (up to 1064 MB/s) and backward compatibility with PCI devices.

PCI Express (PCIe): A newer standard that replaced traditional PCI and PCI-X, offering higher data transfer rates (up to several GB/s) and improved scalability.

Connectors: PCI cards are typically inserted into expansion slots on the motherboard, with various form factors such as PCI, PCI-X, and PCIe slots.



PCIe x 1 is 1 lane PCIe and
PCIe x 16 is 16 lane PCIe.

The term “**lane**” refers to
a set of differential signal
pairs (transmit and
receive) that work
together to transmit data.

PCIe is ***PCI express***

<u>S.NO</u>	<u>PCI</u>	<u>PCI express</u>
1.	PCI is a computer bus that connects hardware devices.	An advanced version of PCI bus that connects graphic ports, Wifi and other devices.
2.	It was introduced in the year 1992 by Intel.	It was introduced in the year 2003 by Intel, Dell and other organizations.
3.	It is a parallel bus interface	It is a serial bus interface.
4.	Conventional PCI is the other name for PCI.	PCI-e is the abbreviated name for PCI Express.
5.	It works slower.	It works faster.
6.	PCI provides a slower data rate	PCI Express provides faster data rate.
7.	The PCI slots are standardized	The PCI-e slots depends upon number of lanes.
8.	The speed of a PCI slot is upto 133MB/s.	The speed of a PCI-e slot is upto 16 GB/s.
9.	It has less features.	It has more features.
10.	It may or may not have hot swapping feature.	It has the hot swapping feature.

SCSI (Small Computer System Interface)

SCSI (Small Computer System Interface) is used to connect and communicate between computers and peripheral devices, such as hard disk drives, tape drives, CD/DVD drives, and scanners. SCSI was originally developed as both a protocol and a parallel physical interface.

SCSI is a set of standards for connecting and transferring data between computers and peripheral devices, such as hard drives, tape drives, and scanners.

It provides a high-performance interface for storage devices.




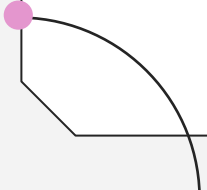
Features

High speed: SCSI offers high-speed data transfer rates, making it suitable for demanding storage applications.

Multiple device support: SCSI supports connecting multiple devices to a single SCSI bus, allowing for efficient use of resources.

Hot swapping: Some SCSI implementations support hot-swapping, enabling devices to be added or removed without shutting down the system.

Wide compatibility: SCSI is compatible with various types of storage devices, including hard drives, tape drives, optical drives, and solid-state drives.



Variants

SCSI-1: The original SCSI standard, offering data transfer rates of up to 5 MB/s.

SCSI-2: An improved version of SCSI-1, offering higher speeds (up to 10 MB/s) and additional features such as disconnect/reconnect and command queuing.

SCSI-3: The latest version of the SCSI standard, offering even higher speeds (up to several GB/s) and improved features for storage networking and data protection.

Connectors: SCSI devices are typically connected to the computer via a SCSI bus, using various types of connectors such as SCSI-1 Centronics, SCSI-2 HD50, SCSI-3 VHDCI, and SCSI-3 SCA.

SCSI

Small Computer Systems Interface



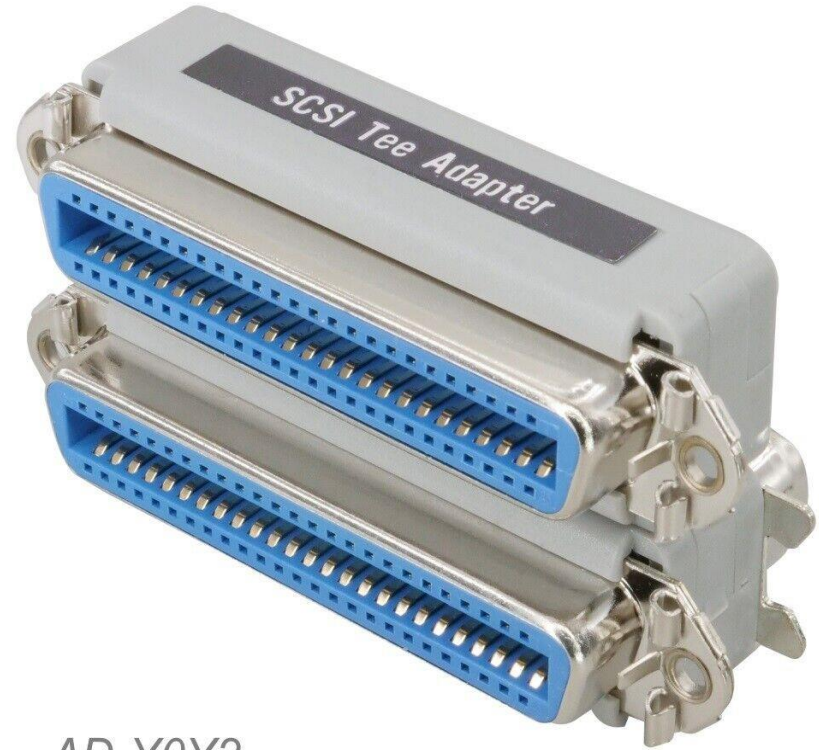
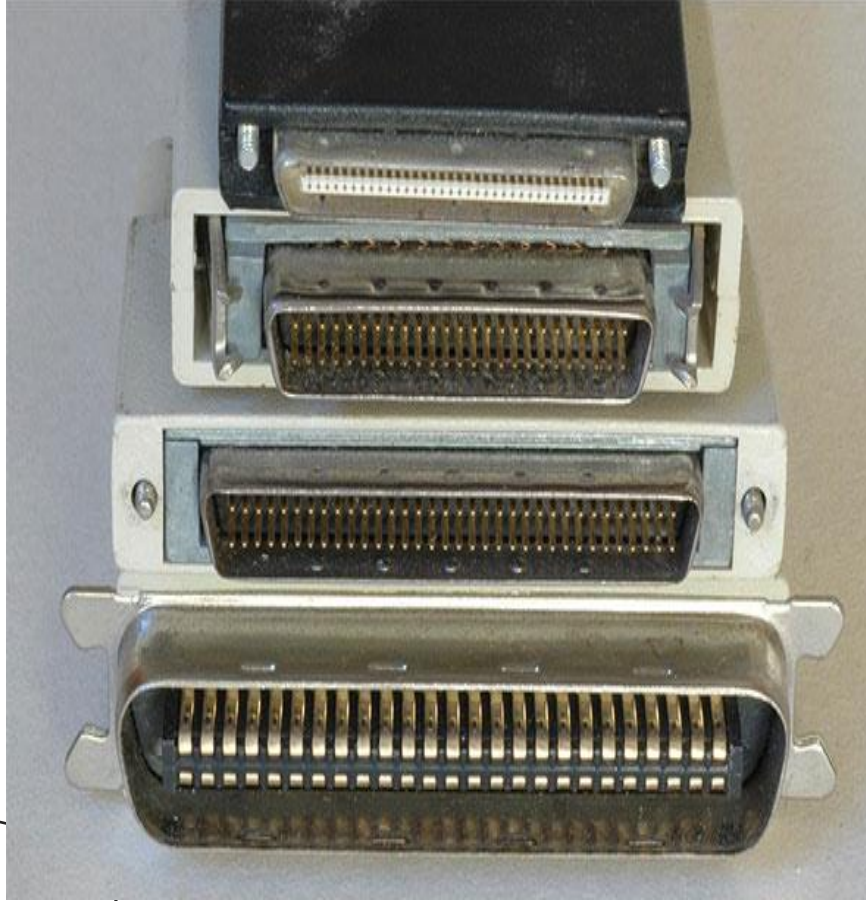
LVD: Low voltage differential



SE: Serial

HVD: High Voltage Differential





AD-Y0Y2

USB (Universal Serial Bus):

USB is a widely used standard for connecting peripheral devices to computers, such as keyboards, mice, printers, and external storage devices.

It provides a versatile and convenient interface for connecting a wide range of devices.



Features

Plug and Play: USB supports Plug and Play functionality, allowing devices to be easily connected and configured without requiring manual driver installation.

Hot swapping: USB devices can be hot-swapped, meaning they can be added or removed from the system without shutting down.

Power supply: USB provides power to connected devices, eliminating the need for separate power adapters in many cases.

Scalability: USB supports multiple device types and can be extended using hubs to connect more devices to a single USB port.





Variants

USB 1.x: The original USB standard, offering data transfer rates of up to 12 Mbps (USB 1.1) or 1.5 Mbps (USB 1.0).

USB 2.0: An improved version of USB 1.x, offering higher speeds (up to 480 Mbps) and improved power management features.

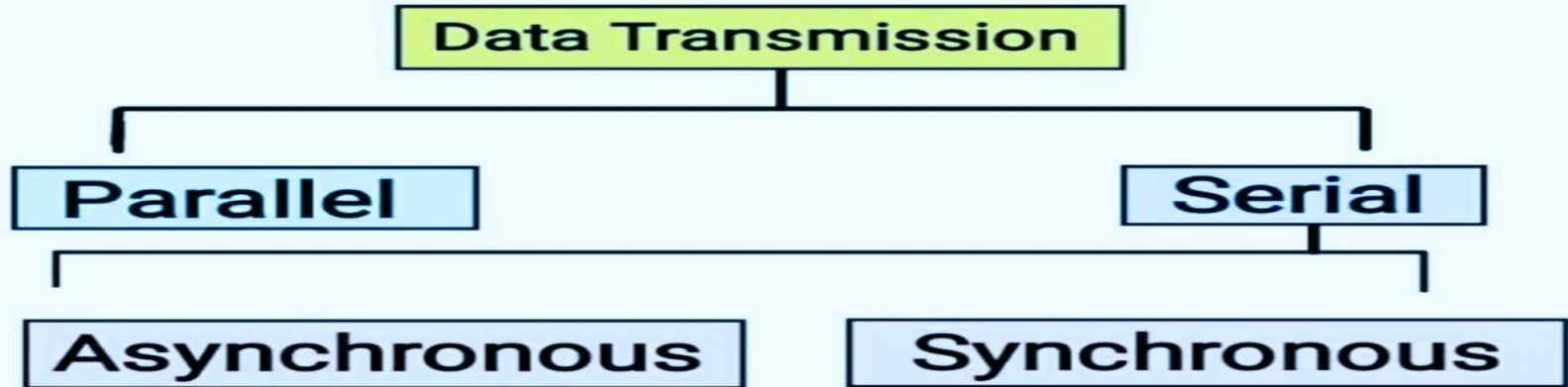
USB 3.x: The latest version of the USB standard, offering even higher speeds (up to several Gbps) and improved power delivery capabilities.

Connectors: USB devices are typically connected to the computer via USB ports, using various types of connectors such as USB-A, USB-B, USB-C, and micro-USB.



Serial and Parallel Transfer of Data

Communication: Interchanging information between individuals in the form of Audio, Video, Written documents etc.


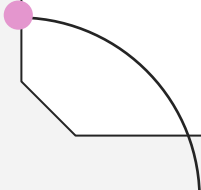


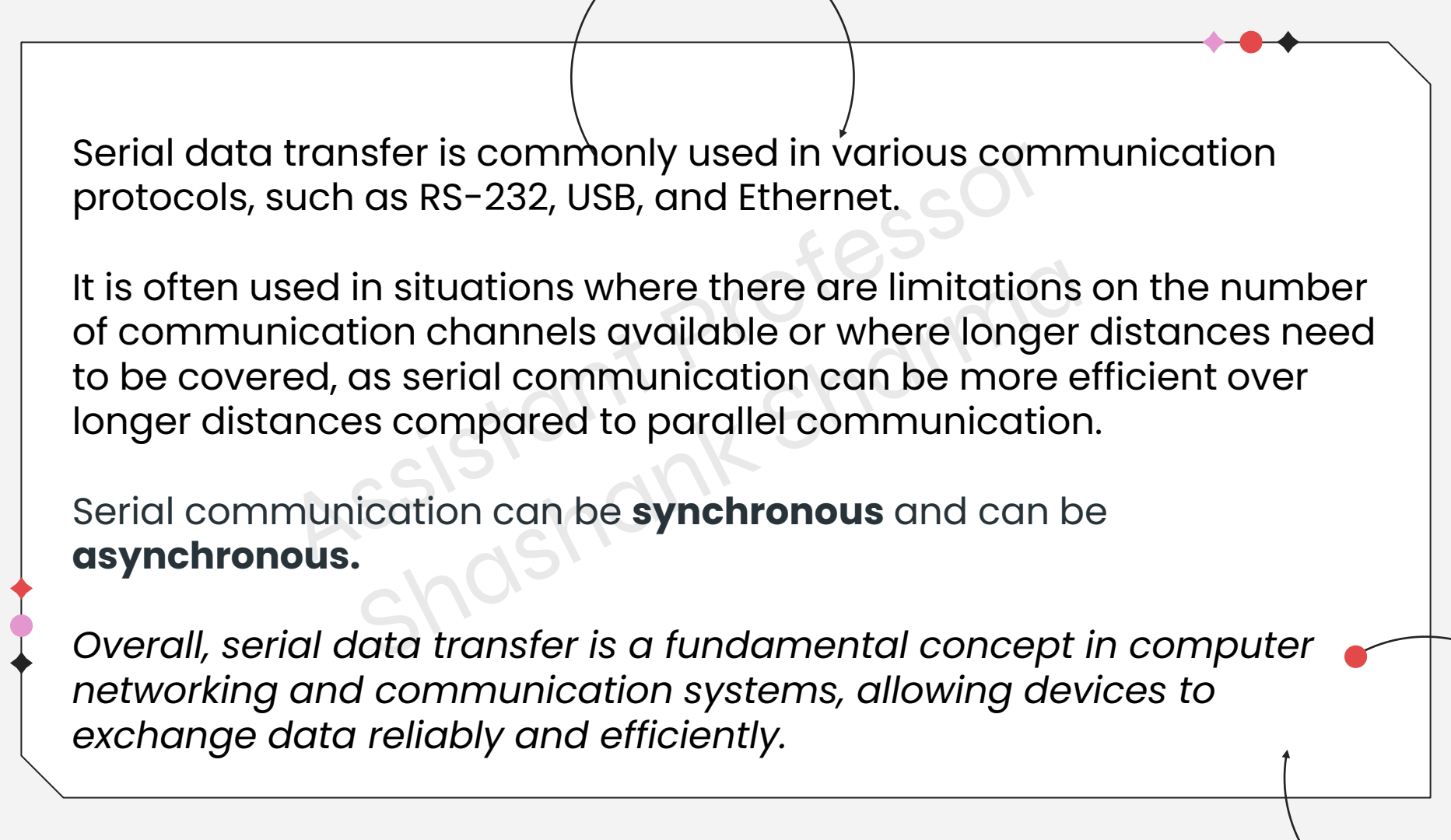


Serial Data transfer

Serial data transfer involves sending data one bit at a time over a single communication channel or wire. This is in contrast to parallel data transfer, where multiple bits are sent simultaneously over multiple channels.

In serial data transfer, the bits are typically sent in a specific order, with each bit following the previous one. This can be done using different encoding schemes, such as ASCII or Unicode for text data, or binary encoding for numerical data.





Serial data transfer is commonly used in various communication protocols, such as RS-232, USB, and Ethernet.

It is often used in situations where there are limitations on the number of communication channels available or where longer distances need to be covered, as serial communication can be more efficient over longer distances compared to parallel communication.

Serial communication can be **synchronous** and can be **asynchronous**.

Overall, serial data transfer is a fundamental concept in computer networking and communication systems, allowing devices to exchange data reliably and efficiently.




Pros:

Simplicity: Serial communication typically requires fewer wires compared to parallel communication, making it simpler to implement and manage.

Longer Distance: Serial communication can often be more reliable over longer distances compared to parallel communication, as it is less susceptible to signal degradation.

Cost-Effective: Serial communication can be more cost-effective, especially for long-distance communication, as it requires fewer wires and components.





Scalability: It's easier to scale serial communication systems, as adding additional devices usually only requires connecting them to the existing communication channel.

Compatibility: Many standard communication protocols, such as USB and Ethernet, use serial data transfer, making it widely compatible with various devices and systems.




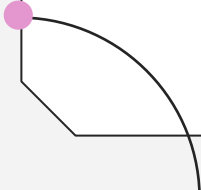


Cons:

Slower Speed: Serial communication typically has slower data transfer rates compared to parallel communication, as it sends data one bit at a time.

Increased Latency: Sending data serially can introduce additional latency, especially when dealing with large amounts of data or when using slower communication protocols.

Complexity for Synchronous Communication: Synchronous serial communication, where data is sent along with a clock signal, can be more complex to implement compared to asynchronous communication.


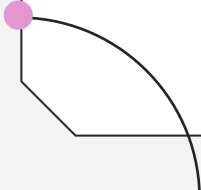


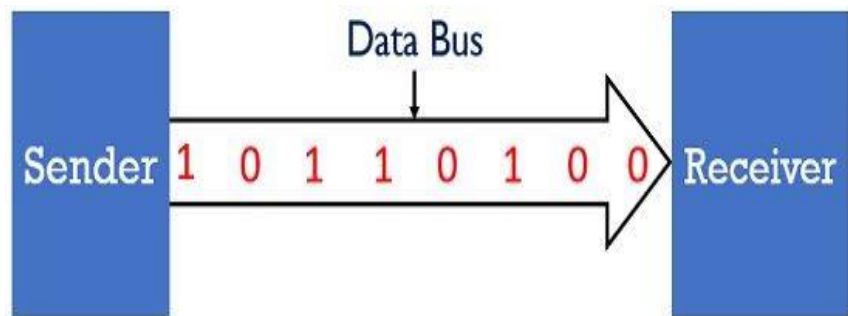


Limited Bandwidth: Serial communication channels have limited bandwidth, which can become a bottleneck when transferring large amounts of data.

Susceptibility to Noise: Because serial communication relies on a single communication channel, it can be more susceptible to noise and interference, especially over longer distances or in noisy environments.

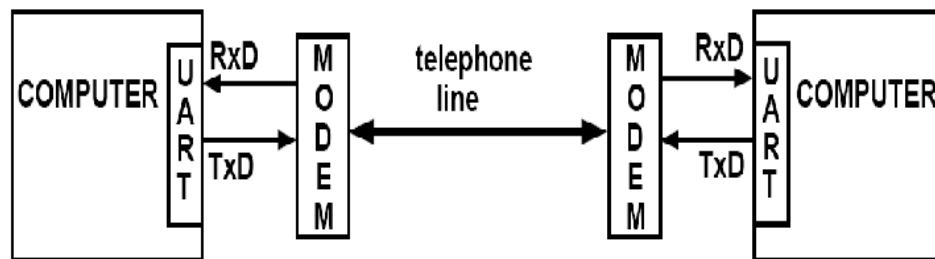
Overall, while serial data transfer offers simplicity, cost-effectiveness, and compatibility, it may not always be the best choice for applications requiring high-speed data transfer or low latency(delay).





Serial Communication

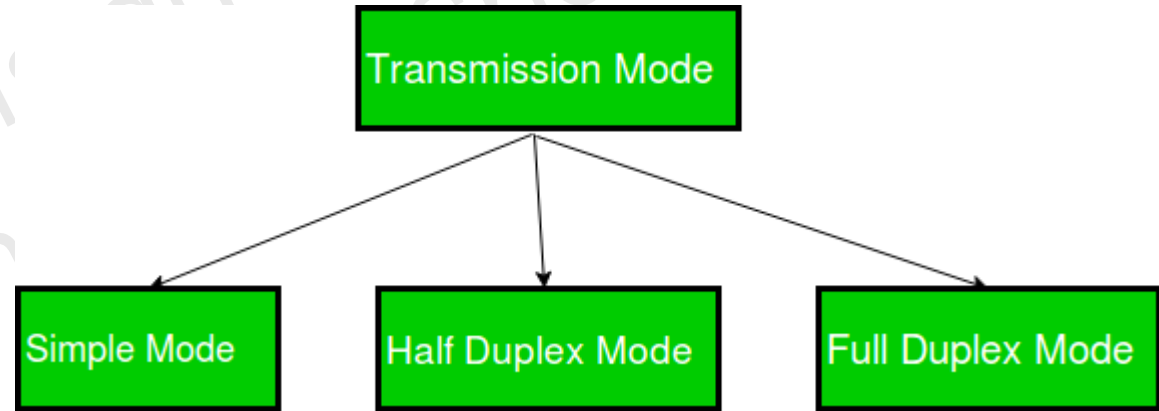
Serial Data Transmission



◆ ● ◆

There are three primary transmission modes in serial communication:

1. Simplex
2. Half-Duplex
3. Full-Duplex



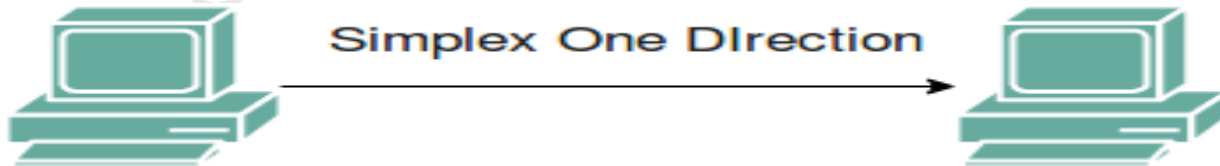
Simplex

In Simplex mode, the communication is unidirectional, as on a one-way street. Only one of the two devices on a link can transmit, the other can only receive. The simplex mode can use the entire capacity of the channel to send data in one direction.

Example: Keyboard and traditional monitors.

The keyboard can only introduce input, the monitor can only give the output.

Also, Television Broadcasting, where TV station transmit signal to viewers without receiving any feedback.



Advantages:

Simplex mode is the easiest and most reliable mode of communication.

It is the most cost-effective mode, as it only requires one communication channel.

There is no need for coordination between the transmitting and receiving devices, which simplifies the communication process.

Simplex mode is particularly useful in situations where feedback or response is not required, such as broadcasting or surveillance.

Disadvantages:

Only one-way communication is possible.

There is no way to verify if the transmitted data has been received correctly.

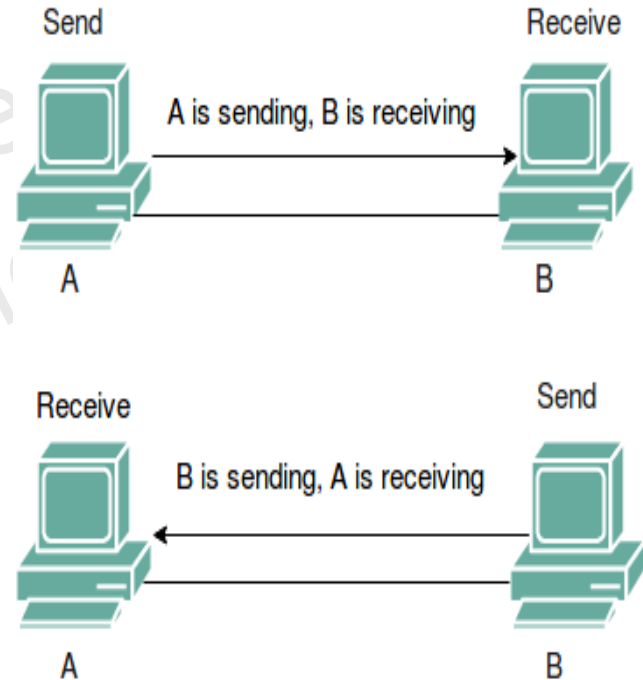
Simplex mode is not suitable for applications that require bidirectional communication.

Half-Duplex Mode

In half-duplex mode, each station can both transmit and receive, but not at the same time. When one device is sending, the other can only receive, and vice versa.

The half-duplex mode is used in cases where there is no need for communication in both directions at the same time. The entire capacity of the channel can be utilized for each direction.

Example: Walkie-talkie in which message is sent one at a time and messages are sent in both directions.



Advantages:

Half-duplex mode allows for bidirectional communication, which is useful in situations where devices need to send and receive data.

It is a more efficient mode of communication than simplex mode, as the channel can be used for both transmission and reception.

Half-duplex mode is less expensive than full-duplex mode, as it only requires one communication channel.

Disadvantages:

Half-duplex mode is less reliable than Full-Duplex mode, as both devices cannot transmit at the same time.

There is a delay between transmission and reception, which can cause problems in some applications.

There is a need for coordination between the transmitting and receiving devices, which can complicate the communication process.

Full-Duplex Mode

In full-duplex mode, both stations can transmit and receive simultaneously. In full_duplex mode, signals going in one direction share the capacity of the link with signals going in another direction, this sharing can occur in two ways:-

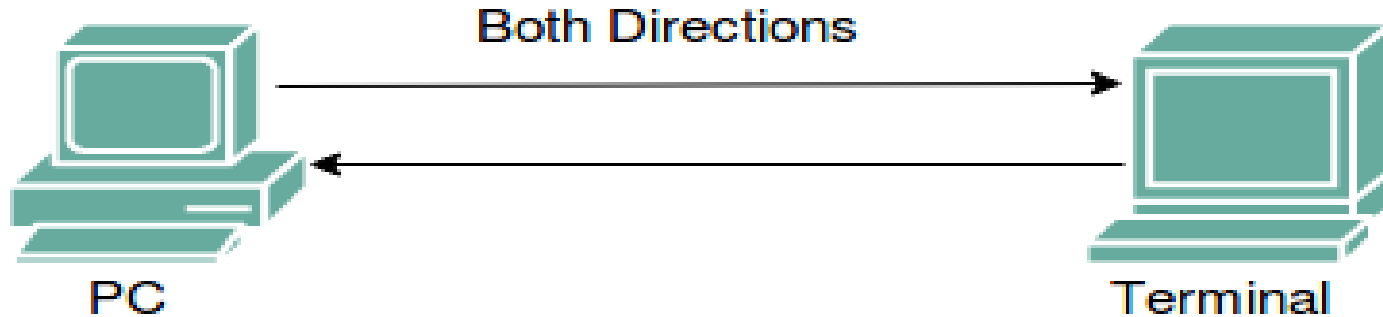
Either the link must contain two physically separate transmission paths, one for sending and the other for receiving.

Or the capacity is divided between signals traveling in both directions.

Full-duplex mode is used when communication in both directions is required all the time. The capacity of the channel, however, must be divided between the two directions.

Full-Duplex Mode

Example: Telephone Network in which there is communication between two persons by a telephone line, through which both can talk and listen at the same time.



Advantages:

Full-duplex mode allows for simultaneous bidirectional communication, which is ideal for real-time applications such as video conferencing or online gaming.

It is the most efficient mode of communication, as both devices can transmit and receive data simultaneously.

Full-duplex mode provides a high level of reliability and accuracy, as there is no need for error correction mechanisms.

Disadvantages:

Full-duplex mode is the most expensive mode, as it requires two communication channels.

It is more complex than simplex and half-duplex modes, as it requires two physically separate transmission paths or a division of channel capacity.

Full-duplex mode may not be suitable for all applications, as it requires a high level of bandwidth and may not be necessary for some types of communication.

There are two types of serial data transfer methods, they are:

Asynchronous Serial Communication: In asynchronous serial communication, data is transmitted without any reference to a clock signal. Instead, each character is preceded by a start bit and followed by one or more stop bits. This method is commonly used in applications where the transmission speed can vary, and synchronization between the transmitter and receiver is achieved through the start and stop bits.

Synchronous Serial Communication: In synchronous serial communication, data is transmitted with a clock signal that synchronizes the timing between the transmitter and receiver. Both the transmitter and receiver share a common clock signal, ensuring that the data is transmitted and received at the same rate. Synchronous serial communication is often faster and more efficient than asynchronous communication but requires a more precise timing mechanism.

Parallel data transfer

Parallel data transfer is a method of transmitting data where multiple bits are sent simultaneously over multiple data lines or channels. Here's a detailed explanation of how parallel data transfer works:

Simultaneous Transmission:

In parallel data transfer, each bit of data is transmitted simultaneously across its own dedicated channel or wire.

For example, in an 8-bit parallel communication system, eight bits of data are transmitted simultaneously over eight data lines.

Data Lines:

The number of data lines determines the number of bits that can be transmitted simultaneously.

More data lines allow for the transmission of a larger number of bits in parallel, increasing the data transfer rate.

Timing:

Parallel data transfer requires precise timing to ensure that data is transmitted and received correctly.

Timing signals may be used to synchronize the transmission and reception of data across multiple channels.

Applications:

Parallel data transfer is commonly used in various applications where high-speed data transfer is required, such as:

- Memory interfaces in computers (e.g., DDR SDRAM)
- Communication buses within electronic systems (e.g., PCI, PCIe)
- Data transfer between microprocessors and peripherals

Advantages:

High Data Transfer Rates: Parallel data transfer allows for higher data transfer rates compared to serial communication, as multiple bits are transmitted simultaneously.

Efficiency: Transmitting multiple bits simultaneously can improve communication efficiency, especially for applications requiring large amounts of data to be transferred quickly.

Limitations:

Complexity: Implementing parallel data transfer systems can be more complex compared to serial communication, especially for systems with a large number of data lines.

Signal Integrity: Maintaining signal integrity becomes challenging as the number of data lines increases, especially at higher data rates and over longer distances.

Generally, parallel communication tends to be synchronous rather than asynchronous. This preference arises from the nature of parallel communication, where multiple bits are transferred simultaneously across multiple data lines or channels.

Synchronous parallel communication involves transferring data across multiple channels while being synchronized by a common clock signal. This synchronization ensures that all bits across the parallel channels are transferred in sync with each other, minimizing timing issues and ensuring data integrity. Many high-speed parallel communication standards, such as **PCI Express (PCIe)**, use synchronous communication.

While **asynchronous parallel communication** is possible, it is **less common and more complex to implement**.

Asynchronous communication typically involves each data byte being accompanied by start and stop bits to delineate the boundaries of each byte. This can introduce timing challenges and may not be as efficient for parallel communication, especially at high data rates.

Parallel Asynchronous Communication Example (Centronics Parallel Port) It was widely used to connect printers and other peripherals to computers before the advent of USB.

So, while both synchronous and asynchronous parallel communication are technically feasible, synchronous communication is more prevalent and preferred in most cases due to its simpler implementation and better performance for parallel data transfer.

Synchronous & Asynchronous mode of data transfer:

Data transfer can be categorized into two main modes: ***synchronous and asynchronous***.



Synchronous Data Transfer

Imagine a synchronized dance performance. The dancers (data) move in perfect unison following a common beat (clock signal). This is similar to synchronous data transfer.

Data Transmission: Data is sent in fixed-sized blocks or frames.

Transmission Method: Relies on a shared clock signal between sender and receiver to maintain timing accuracy.

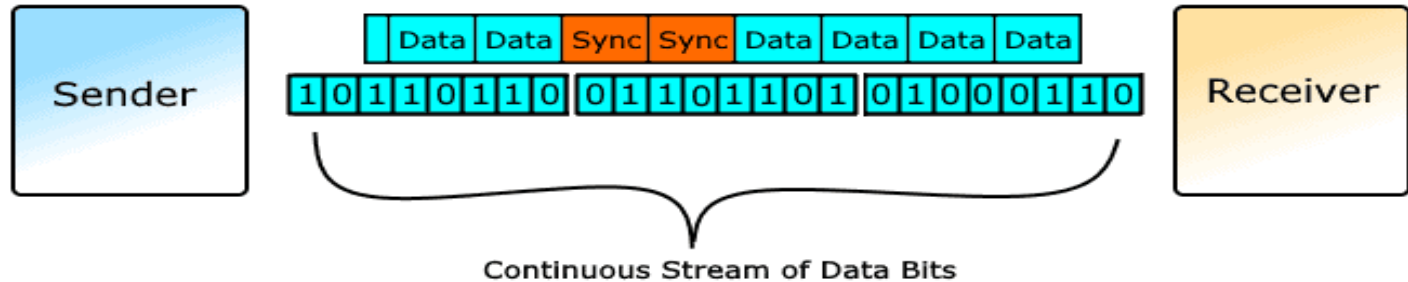
Speed: Generally faster due to streamlined transmission without waiting for individual characters.

Complexity: Requires complex hardware and software for synchronization

Error Detection: Errors can be detected and corrected in real-time during transmission.

Cost: More expensive due to the complexity involved.

Example: Streaming a high-definition video. The video data is broken down into frames, and both the sender (streaming service) and receiver (your device) use a common clock to ensure smooth, uninterrupted playback.



Synchronous Transmission

Asynchronous Data Transfer

Think of sending individual text messages. Each message (data) is independent and doesn't require strict timing coordination. This is analogous to asynchronous data transfer.

Data Transmission: Data is sent one byte or character at a time.

Transmission Method: Doesn't require a shared clock signal. Each byte includes start and stop bits for identification.

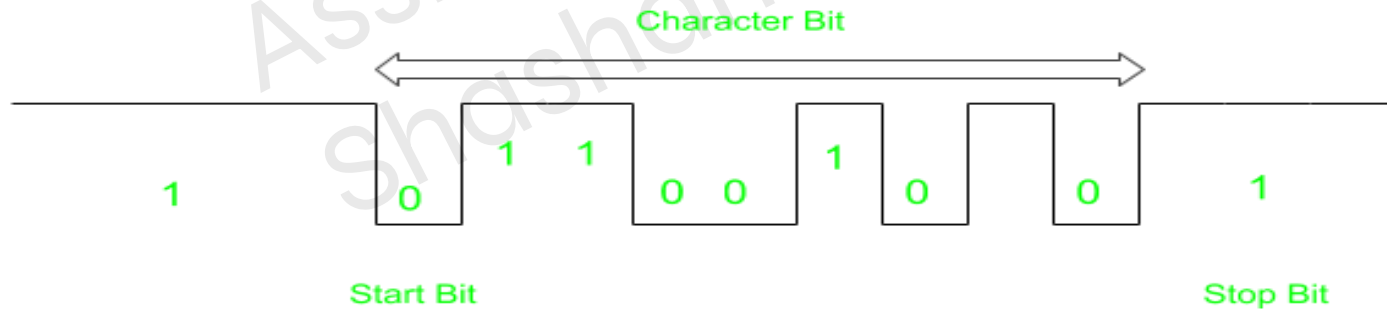
Speed: Slower than synchronous due to the extra overhead of start/stop bits.

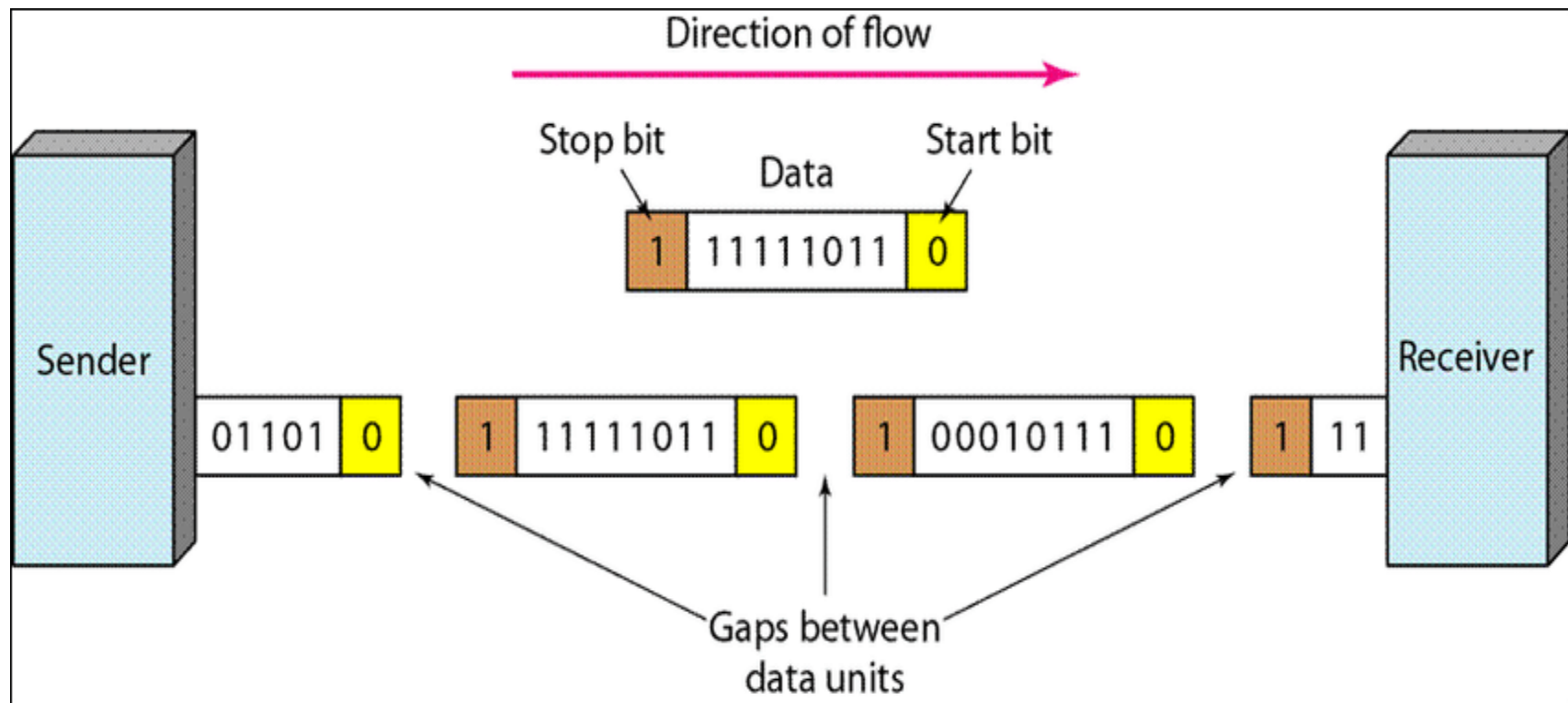
Complexity: Simpler to implement using hardware.

Error Detection: Errors are typically detected upon receiving the entire data stream.

Cost: More economical due to simpler hardware requirements.

Example: Downloading a file from the internet. Each data packet containing a portion of the file is sent independently with start and stop bits, allowing for reliable transfer even with slight delays or interruptions.





Choosing the Right Mode:

The choice between synchronous and asynchronous data transfer depends on the specific application:

Synchronous: Ideal for high-speed, real-time data transfer with minimal errors, like streaming videos or audio calls.

Asynchronous: Well-suited for applications where occasional delays are acceptable, like downloading files or sending emails.

Data transfer between the CPU and I/O devices

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

- **Programmed I/O**
- **Interrupt initiated I/O**
- **Direct Memory Access (DMA)**

Programmed I/O

Programmed I/O, or Program-Controlled I/O, is a method of input/output operation where the CPU controls the data transfer between the CPU and an external device. In programmed I/O, the CPU actively manages the data transfer process by executing specific instructions to read from or write to the I/O device.

Here's how programmed I/O works:

Initialization: The CPU initializes the I/O operation by setting up the necessary registers and flags to communicate with the I/O device.

Data Transfer: The CPU executes I/O instructions (e.g., IN and OUT instructions on x86 architecture) to transfer data between the CPU and the I/O device. These instructions typically involve specifying the I/O device's address and the data to be transferred.

Polling: After initiating the data transfer, the CPU typically polls the status of the I/O device to check if the transfer is complete. This involves repeatedly checking a status register or flag to determine if the device is ready to send or receive data.

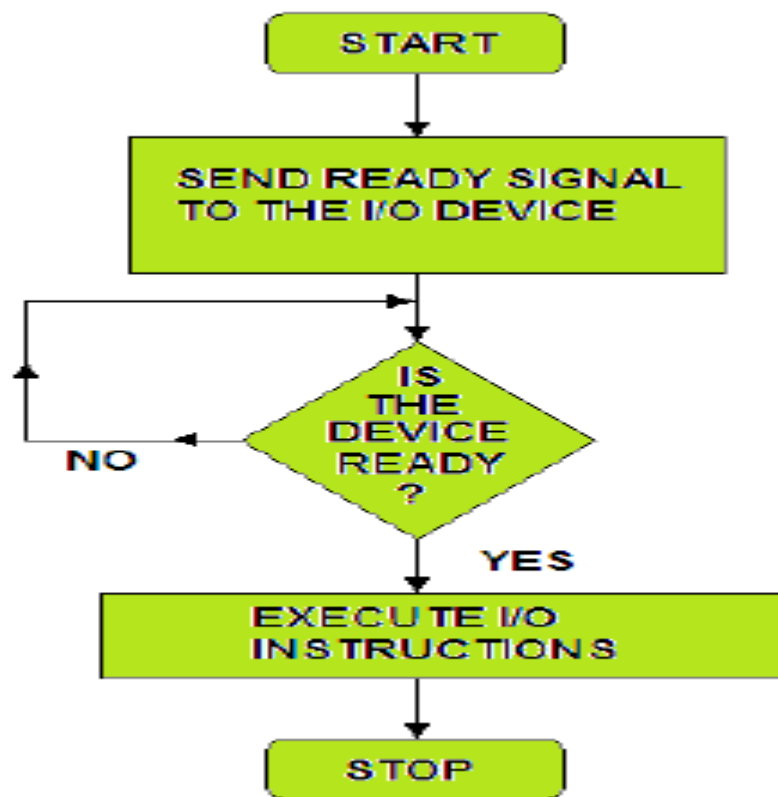
Data Processing: Once the data transfer is complete, the CPU processes the received data or continues with other tasks as needed.

Programmed I/O is straightforward and easy to implement, as it relies on the CPU to manage the entire data transfer process.

However, it can be inefficient and CPU-intensive, especially for slow I/O devices or when large amounts of data need to be transferred. This is because the CPU must actively participate in every data transfer operation, which can consume significant CPU cycles and slow down overall system performance.

In modern computer systems, programmed I/O is often used for simple I/O operations or when direct CPU control is necessary.

However, more efficient I/O techniques such as interrupt-driven I/O and DMA (Direct Memory Access) are commonly used for high-speed and bulk data transfer operations. These techniques offload the data transfer process from the CPU, allowing it to focus on other tasks while data is transferred in the background.



Programmed I/O

Interrupt-initiated I/O

Interrupt-initiated I/O, also known as Interrupt-driven I/O, is a method of input/output (I/O) operation where the CPU initiates I/O operations and then continues with other tasks while waiting for the completion of the I/O operation. Instead of continuously polling the status of the I/O device, the CPU is interrupted by the device when the I/O operation is complete or when the device requires attention.

Here's how interrupt-initiated I/O works:

Initialization: The CPU initializes the I/O operation by setting up the necessary registers and flags to communicate with the I/O device.

Start I/O Operation: The CPU initiates the I/O operation by issuing a command to the I/O device and then continues executing other instructions while waiting for the completion of the operation.

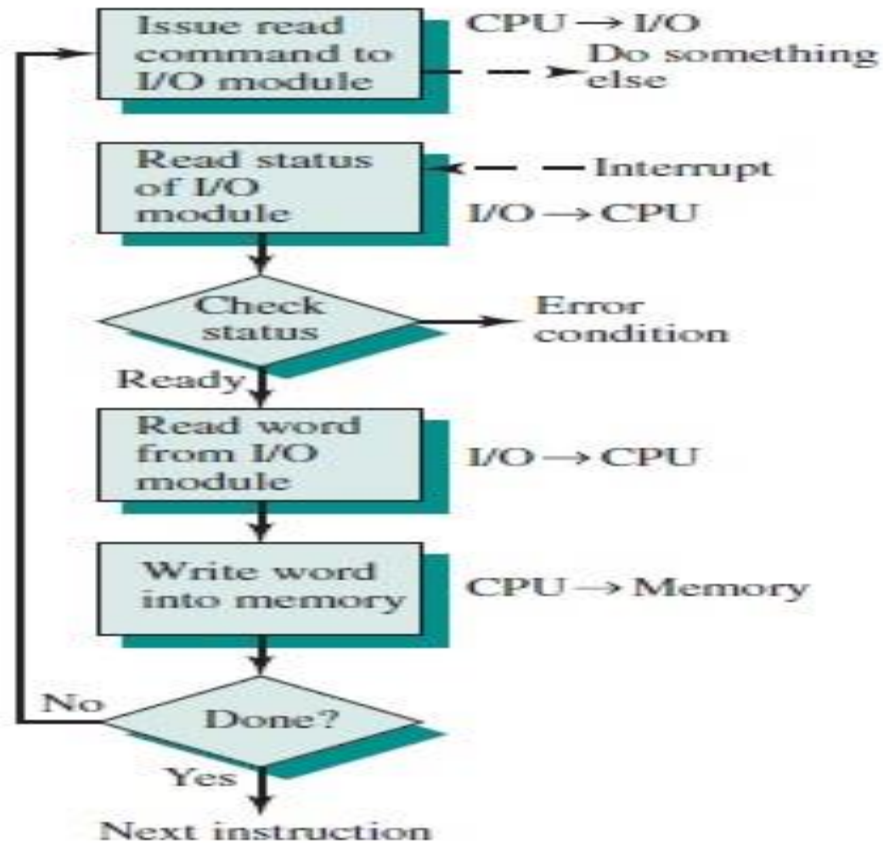
Interrupt Handling: When the I/O operation is complete or when the device requires attention (e.g., an error condition or data ready), the I/O device interrupts the CPU by asserting an interrupt signal.

Interrupt Service Routine (ISR): Upon receiving the interrupt, the CPU suspends its current task and transfers control to an Interrupt Service Routine (ISR) associated with the specific interrupt. The ISR handles the interrupt by servicing the I/O device, which may involve transferring data between the CPU and the device or handling error conditions.

Resume Execution: After servicing the interrupt, the CPU resumes the interrupted task or execution point. If the I/O operation is complete, the CPU can process the received data or continue with other tasks as needed.

Interrupt-initiated I/O is more efficient than programmed I/O because it allows the CPU to perform other tasks while waiting for I/O operations to complete. This improves overall system performance by reducing CPU idle time and minimizing the need for continuous polling of I/O devices.

Additionally, interrupt-driven I/O is well-suited for handling asynchronous events and real-time requirements in computer systems. Interrupt-initiated I/O is commonly used in modern computer systems and is supported by most operating systems and hardware platforms. It is particularly beneficial for high-speed I/O operations and scenarios where multiple I/O devices are present in the system.



Interrupt initiated I/O

Interrupts

Interrupts are signals sent by hardware or software to the CPU to temporarily suspend its current execution and handle a specific event. They allow the CPU to respond to events asynchronously, such as user inputs, hardware errors, or timer expirations, without constantly polling for them.

Interrupts are crucial for multitasking and real-time processing in computer systems. When an interrupt occurs, the CPU saves its current state, switches to a predefined interrupt handling routine, processes the interrupt, and then resumes the previous task.

Interrupts are like sudden interruptions or taps on the shoulder that your computer gets to handle important tasks or events. They're signals from hardware or software that tell the CPU, "Hey, pay attention to this right now!"

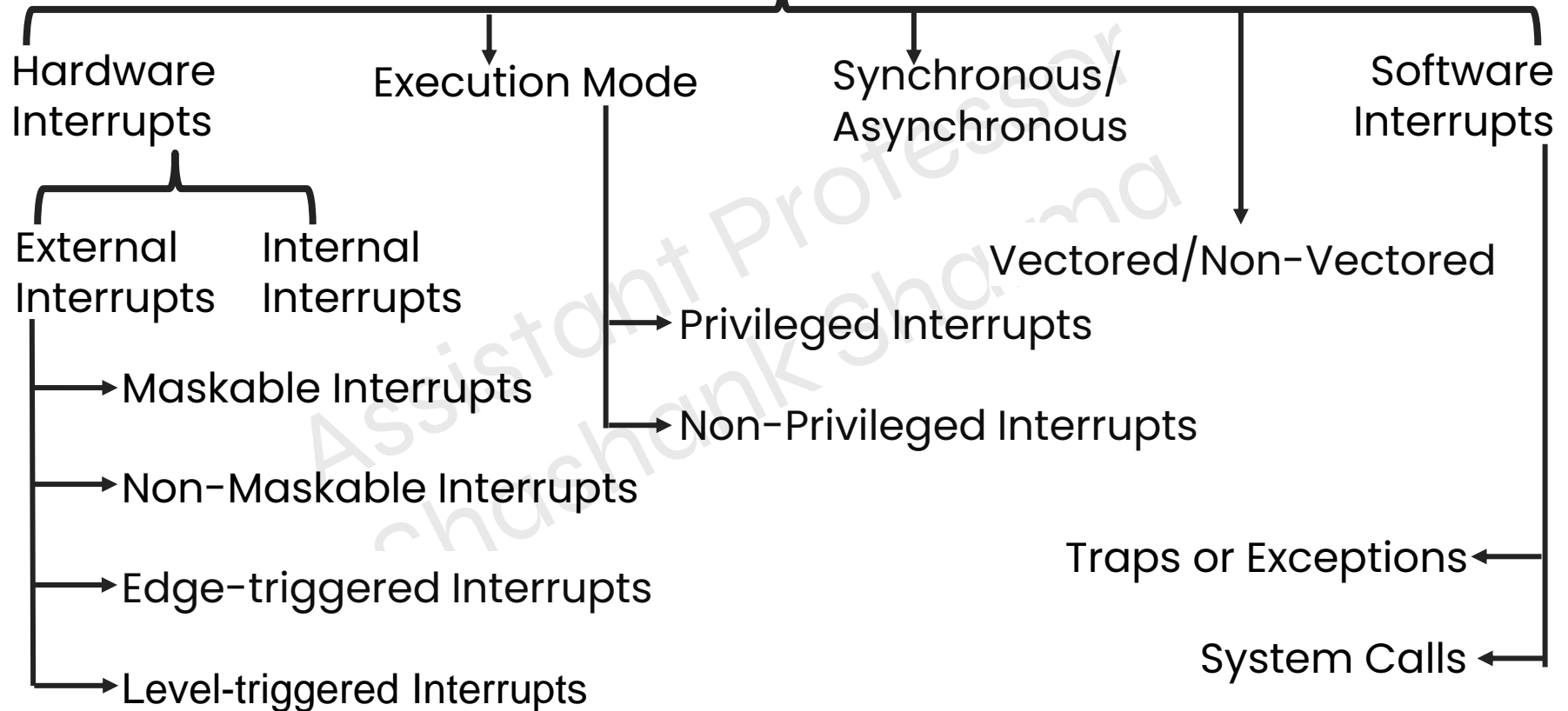
Here's a simple example:

Imagine you're working on your computer, typing a document. Suddenly, your friend sends you an urgent message on chat. Your computer receives this message and interrupts your typing to show you the message. Once you've read and responded to the message, you can go back to typing your document.

In this example:

- Your typing is your CPU working on a task.
- The urgent message is the interrupt, signaling the CPU to switch its attention to the new task (showing the message).
- After dealing with the interrupt (reading and responding to the message), the CPU goes back to the original task (typing the document).

Types of Interrupts



Hardware interrupts

Hardware interrupts are signals sent by external hardware devices to the CPU to request immediate attention or action. These interrupts are used to notify the CPU of events occurring outside the CPU's normal operation, such as user input, data arrival from peripherals, or hardware errors.

In simpler terms, hardware interrupts are like notifications that hardware devices send to the CPU, saying, "Hey, I need your attention right now!"

Here's a basic example:

Imagine you're playing a video game on your computer, and you press a key on your keyboard to make your character jump. When you press the key, the keyboard sends a hardware interrupt signal to the CPU, telling it that a key has been pressed. The CPU then pauses the game momentarily, processes the interrupt, and jumps your character accordingly. After handling the interrupt, the CPU resumes running the game as usual.

In this example:

- Pressing the key on the keyboard generates a hardware interrupt.
- The CPU stops what it's doing, processes the interrupt, and responds to the input (making the character jump).

- Once the interrupt is handled, the CPU goes back to running the game.

Hardware interrupts are essential for allowing external devices to communicate with the CPU and ensuring that the computer can respond to events in real-time.

1. External Interrupts:

External interrupts are signals generated by external hardware devices to request attention from the CPU. These interrupts originate from outside the CPU and are typically triggered by events such as user input, data arrival from peripherals, or hardware errors.

a) Maskable Interrupts Vs Non Maskable Interrupts

Maskable interrupts:

Maskable interrupts are interrupts that can be temporarily ignored or delayed by the CPU based on its current state. They are typically used for non-critical events, allowing the CPU to prioritize its tasks and handle the interrupt when it's convenient.

Example: Consider a scenario where a user is typing a document on their computer. While typing, the computer receives a signal from the network card indicating that new data has arrived. This signal is an interrupt, but since it's maskable, the CPU can choose to delay handling it until the user finishes typing or until it's more convenient to process. The CPU might buffer the incoming network data and handle the interrupt later when the system is idle.

Non-Maskable Interrupts (NMI):

Non-Maskable Interrupts (NMIs) are interrupts that cannot be ignored or delayed by the CPU. They are usually reserved for critical events that require immediate attention, such as hardware failures or power outages. NMIs take precedence over all other interrupts and force the CPU to stop its current operation and handle the emergency situation immediately.

Example: Imagine a computer system encounters a critical hardware error, such as a memory parity error or a sudden loss of power. In such cases, the affected hardware component sends a non-maskable interrupt (NMI) signal to the CPU. The CPU immediately stops its current operation and switches to a predefined NMI handler routine to handle the emergency situation. This allows the system to take appropriate actions to prevent data corruption or system damage.

b) Edge-Triggered Interrupts Vs Level-Triggered Interrupts

Edge-triggered interrupts

Edge-triggered interrupts are interrupts that are triggered by a change in the signal's edge, either rising or falling. These interrupts respond to the transition of the signal from one state to another, rather than the signal's continuous level.

Example: Consider a scenario where a microcontroller receives interrupts from a sensor that detects motion. The sensor outputs a digital signal that changes from low to high (rising edge) when motion is detected and from high to low (falling edge) when motion stops. The microcontroller is configured to trigger an interrupt whenever the sensor's signal transitions from low to high. This edge-triggered interrupt allows the microcontroller to respond immediately when motion is detected without continuously monitoring the sensor's signal.

Level-Triggered Interrupts:

Level-triggered interrupts are interrupts that are triggered when the signal remains at a certain level for a sustained period. These interrupts respond to the continuous state of the signal, rather than its transition.

Example: Imagine a scenario where a microcontroller receives interrupts from a temperature sensor. The sensor outputs a digital signal that remains high when the temperature exceeds a certain threshold and low when it falls below the threshold. The microcontroller is configured to trigger an interrupt whenever the sensor's signal remains high for a sustained period, indicating that the temperature has exceeded the threshold. This level-triggered interrupt allows the microcontroller to respond to the elevated temperature condition and take appropriate action, such as activating a cooling system or sending an alert.

2. Internal Interrupts:

Internal interrupts are interrupts generated by internal hardware components of a computer system, such as timers, counters, or error detection circuits. These interrupts are not triggered by external devices but rather by events occurring within the CPU or other internal components.

Example: Consider a scenario where a microcontroller includes a timer module. This timer module generates an internal interrupt signal when a predefined time period elapses. The CPU can use this internal interrupt to perform periodic tasks, such as updating system time, scheduling tasks, or triggering specific operations at regular intervals.

In this example:

- The timer module within the microcontroller generates an internal interrupt signal.
- The CPU responds to this interrupt by temporarily suspending its current operation and executing an interrupt service routine (ISR) associated with the timer interrupt.
- The ISR performs the necessary tasks, such as updating system time or executing scheduled tasks, before allowing the CPU to resume its previous operation.

Internal interrupts are essential for coordinating various operations within the CPU and ensuring timely execution of tasks in a computer system.

Execution Mode Interrupts

In computer systems, execution mode interrupts are a mechanism that allows the processor to temporarily suspend the execution of the current program and switch to handle a more critical event. This event, called an interrupt, can originate from either hardware or software.

There are two main types of execution mode interrupts, distinguished by the context in which they occur:

1.Privileged Interrupts

2.Non-Privileged Interrupts (Software Interrupts or Traps)

1.Privileged Interrupts

These interrupts are triggered by hardware devices or system events that require access to protected resources or instructions reserved for the operating system. Only the operating system kernel, which runs in a privileged mode (also called kernel mode or supervisor mode), can handle these interrupts.

When a privileged interrupt happens, the processor:

- Saves the state of the currently running program (user mode program). This includes the program counter (PC), registers, and other relevant information.
- Switches the processor to kernel mode.
- Executes the Interrupt Service Routine (ISR) designed for that specific interrupt.
- After the ISR finishes, the processor restores the saved state of the user program and resumes its execution.

2.Non-Privileged Interrupts (Software Interrupts or Traps):

These interrupts are generated by software instructions executed by user-mode programs. They are typically used to request operating system services or signal errors. While user-mode programs can initiate non-privileged interrupts, the operating system ultimately handles them in kernel mode.

When a non-privileged interrupt occurs:

- The processor may or may not save the state of the user program (depending on the specific interrupt).
- It switches the processor to kernel mode.
- The operating system's kernel code takes over and handles the interrupt using appropriate routines.
- Upon completion, the operating system might resume the user program if its state was saved or perform some action based on the interrupt type (e.g., terminate the program).

Synchronous and asynchronous interrupts

Synchronous and asynchronous interrupts are two ways interrupts can happen in a computer system, both related to execution mode interrupts (privileged and non-privileged). But they differ in how predictable the timing of the interrupt is:

Synchronous Interrupts (Predictable Timing):

Imagine you're the CPU (central processing unit), the brain of the computer. You're busy running a program, following its instructions one by one. Synchronous interrupts are like a well-behaved student raising their hand in class.

How it works: The CPU itself generates these interrupts based on specific points in the program's instructions. It's like the student knowing exactly when to ask a question because it's relevant to the current topic.

Example: A timer interrupt might occur every millisecond (1/1000th of a second). The CPU knows this is coming and pauses the program at that exact time to handle the timer event.

Asynchronous Interrupts (Unpredictable Timing):

Now imagine you're still the CPU, but this time, someone outside the classroom (like a delivery person) knocks on the door. Asynchronous interrupts are like unexpected events that can happen anytime.

How it works: These interrupts come from external devices or events that occur independently of the program's execution. It's like the delivery person knocking at any point during the class, interrupting the flow.

Examples:

A key press on the keyboard: The keyboard doesn't care when you're in the middle of a program instruction; it sends a signal when you press a key.

A network card receiving data: The network card doesn't wait for the program to be ready; it interrupts when data arrives.

Vectored and Non-Vectored Interrupts

Vectored and non-vectored interrupts are two methods for the processor to identify the source of an interrupt and determine the appropriate code to execute in response. The key difference lies in how the source of the interrupt is communicated to the processor.

Vectored Interrupts

Vectored interrupts are a more efficient way to handle interrupts. Here's how it works:

Interrupt and Vector: When a device (e.g., keyboard, timer) needs attention, it sends an interrupt signal to the processor. However, along with the generic interrupt signal, it also sends a unique identifier called a **vector**. This vector acts like an address or code that tells the processor exactly which device triggered the interrupt.

Vector Table: The processor maintains a special table in memory called the Interrupt Vector Table (IVT). This table stores the starting addresses of the Interrupt Service Routines (ISRs) for each device or event that can generate an interrupt.

Identifying the Source: Using the received vector, the processor can directly look up the corresponding ISR address in the IVT. This eliminates the need to search through a list of all possible devices to find the source.

Executing the ISR: Once the ISR address is found, the processor jumps to that location in memory and starts executing the code specific to handling that particular interrupt.

Benefits of Vectored Interrupts:

Faster Response: Since the processor knows exactly which device caused the interrupt from the vector, it can directly jump to the relevant ISR, leading to a faster response time.

More Organized Code: Having separate ISRs for different devices keeps the interrupt handling code organized and easier to manage.

Example:

- A key press on the keyboard triggers an interrupt.
- Along with the interrupt signal, the keyboard sends a vector (e.g., vector number 1) that identifies it as the source.

- The processor looks up vector number 1 in the IVT, finding the starting address of the keyboard ISR.
- The processor jumps to that address and executes the code specifically designed to handle keyboard input.

Non-Vectored Interrupts

Non-vectorized interrupts, while less common in modern systems, are a simpler approach:

Generic Interrupt Signal: When a device needs attention, it sends a generic interrupt signal to the processor. However, there's no additional information about the source.

Interrupt Handling Routine: The processor has a single interrupt handling routine (IHR) responsible for identifying the source and handling the interrupt.

Identifying the Source (Polling): The IHR typically uses a polling mechanism to scan through a list of all possible devices to determine which one caused the interrupt. This can involve checking individual status registers of each device.

Executing the Code: Once the source is identified, the IHR can either directly handle the interrupt or jump to a specific ISR for that device (if such a mechanism exists within the IHR).

Drawbacks of Non-Vectored Interrupts:

Slower Response: The processor needs to spend time polling devices to identify the source, delaying the actual handling of the interrupt.

Less Organized Code: Interrupt handling code can become cluttered as all devices might be handled within a single routine.

Example:

A timer or network card triggers an interrupt.

The processor receives a generic interrupt signal without any information about the source.

The IHR starts scanning status registers of various devices (e.g., timer, keyboard, network card) to see which one caused the interrupt.

Once the source is identified (e.g., timer), the IHR might directly handle the timer interrupt or jump to a specific timer ISR (if present) to handle the event.

Assistant Professor
Shashank Sharma

software interrupt

A software interrupt is a signal generated by software itself, instructing the processor to temporarily halt the current task and focus on a specific event. It's a way for programs to ask the operating system for help or report issues.

There aren't exactly different types of software interrupts, but there are various reasons why they might be triggered. Here are some common scenarios:

System Calls: When a program needs to perform an action that requires operating system privileges, like accessing a file or communicating with a device, it triggers a software interrupt to call upon the relevant part of the operating system.

Error Handling: If a program encounters an error, it can use a software interrupt to notify the operating system and potentially initiate error handling routines.

Debugging: Debuggers can use software interrupts to pause a program's execution at specific points to examine its state and identify issues. Overall, software interrupts are a fundamental mechanism for communication and control between programs and the operating system, ensuring smooth operation and efficient resource management.

Direct Memory Access

DMA (Direct memory access) is the special feature within the computer system that transfers the data between memory and peripheral devices (like hard drives) without the intervention of the CPU.

Computers avoid burdening the CPU so, they shift the work to a Direct Memory Access controller. Let's see the workings of this in detail.

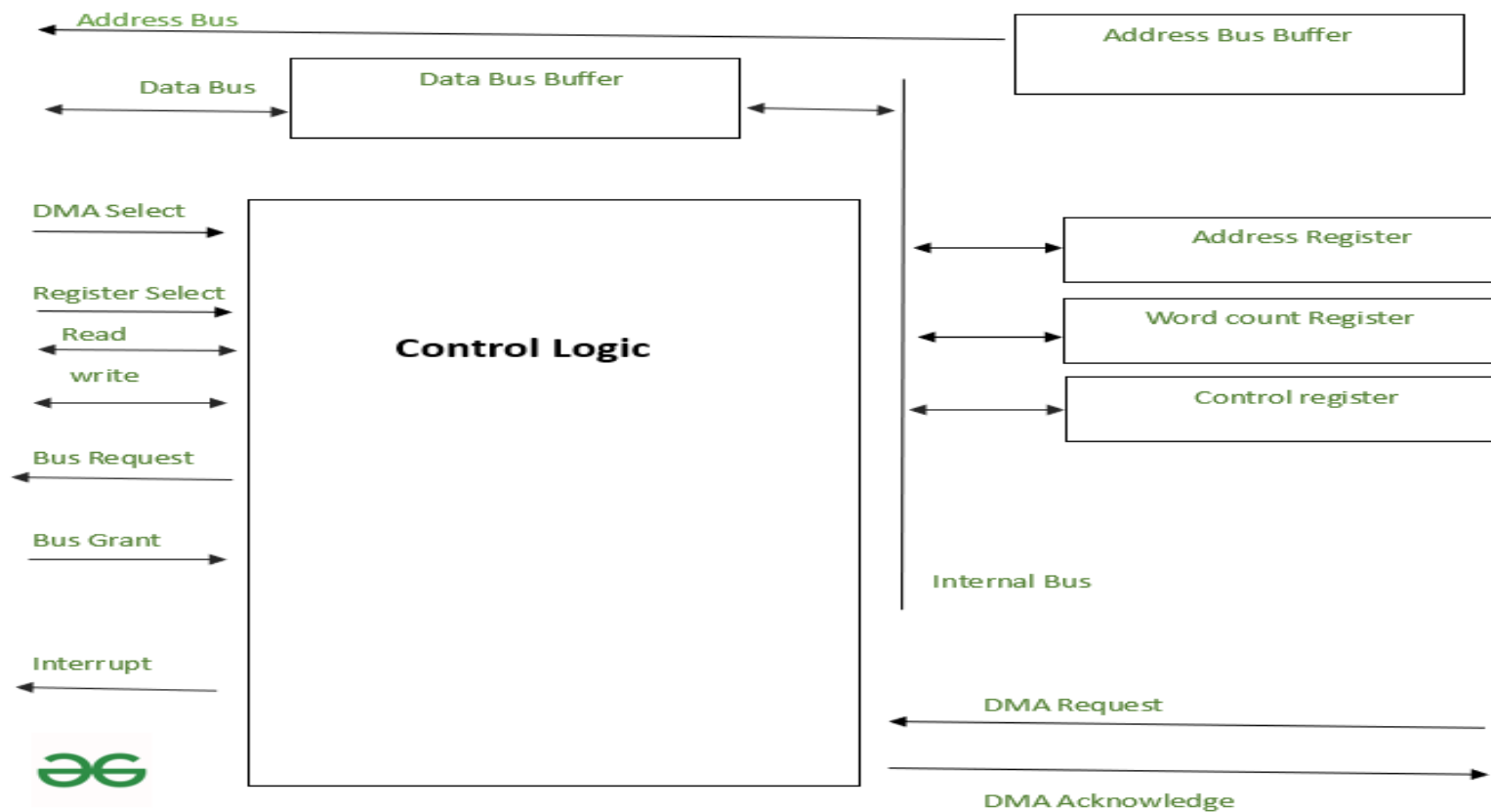
To initiate the DMA transfer the host writes a DMA command block into the memory. This block contains the pointer to the source of the transfer, the pointer to the destination of the transfer, and the count of the number of bytes to be transferred.

This command block can be more complex which includes the list of sources and destination addresses that are not contiguous.

CPU writes the address of this command block and goes to other work.

DMA controller proceeds to operate the memory bus directly, placing the address on it without the intervention of the main CPU.

Nowadays simple DMA controller is a standard component in all modern computers.



Direct Memory Access (DMA) Controller

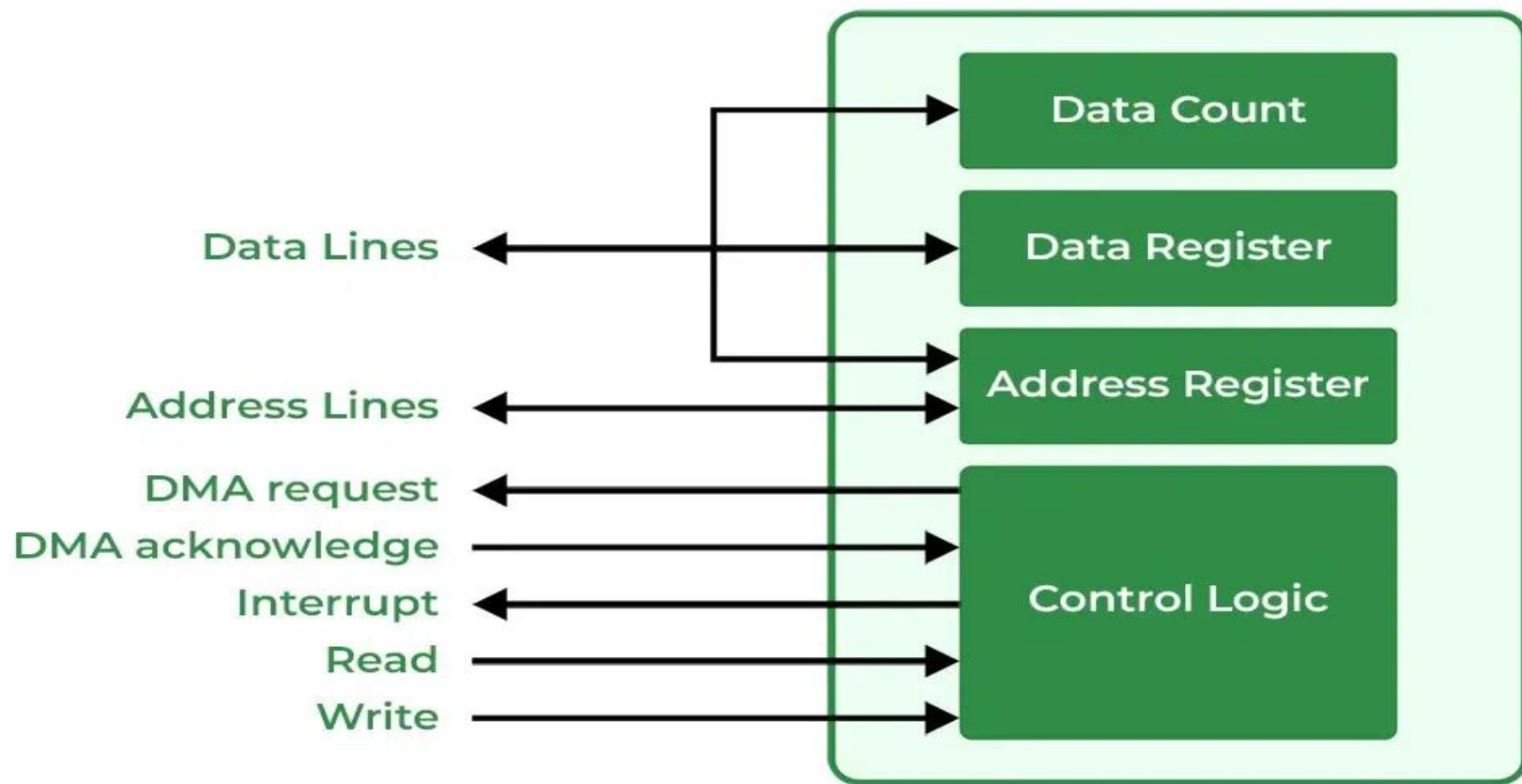
DMA Controller is a hardware device that allows I/O devices to directly access memory with less participation of the processor. DMA controller needs the same old circuits of an interface to communicate with the CPU and Input/Output devices.

What is a DMA Controller?

Direct Memory Access uses hardware for accessing the memory, that hardware is called a DMA Controller. It has the work of transferring the data between Input Output devices and main memory with very less interaction with the processor. The direct Memory Access Controller is a control unit, which has the work of transferring data.

DMA Controller Diagram in Computer Architecture

DMA Controller is a type of control unit that works as an interface for the data bus and the I/O Devices. As mentioned, DMA Controller has the work of transferring the data without the intervention of the processors, processors can control the data transfer. DMA Controller also contains an address unit, which generates the address and selects an I/O device for the transfer of data. Here we are showing the block diagram of the DMA Controller.

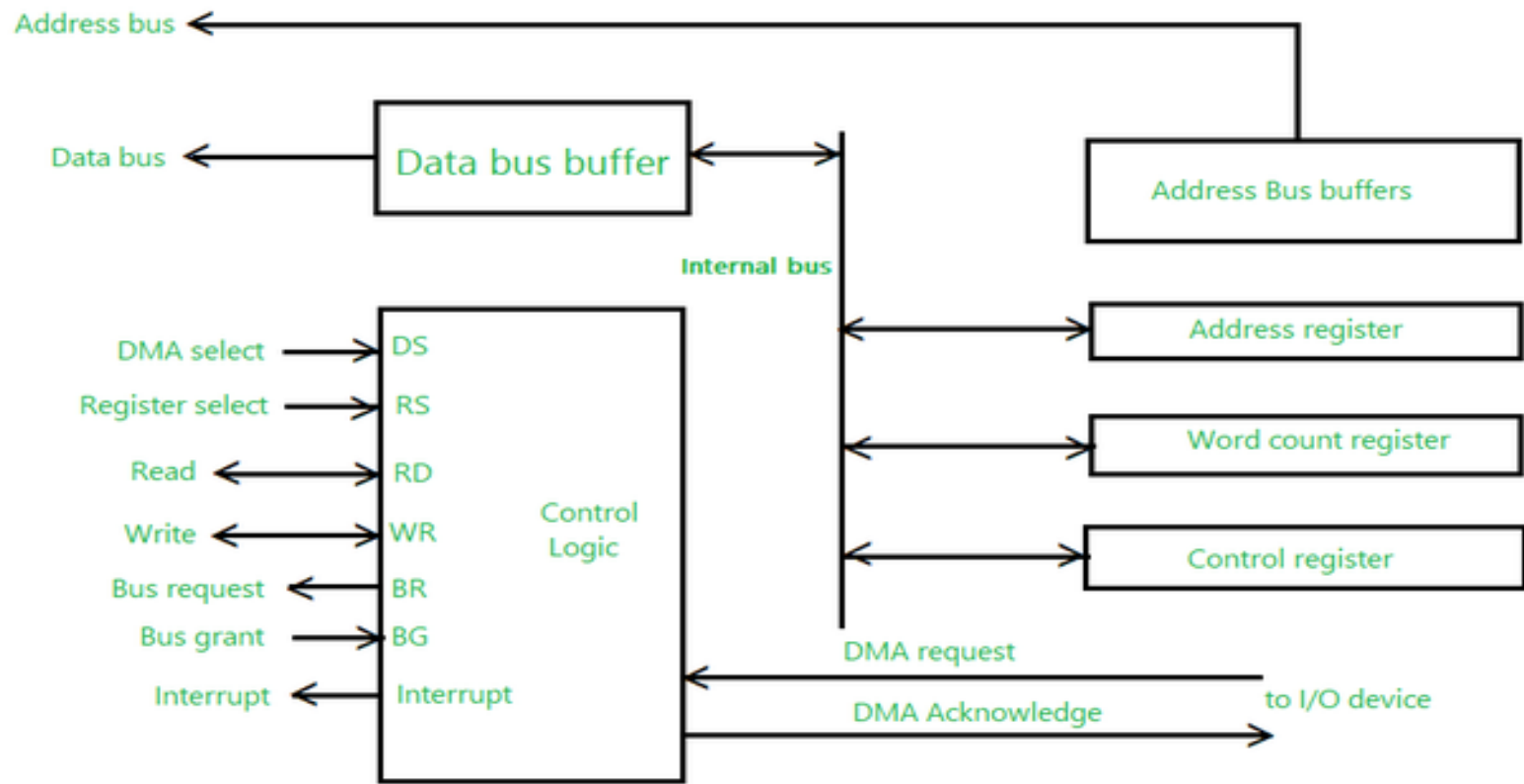


Types of Direct Memory Access (DMA) Controller

There are four popular types of DMA Controller.

- Single-Ended DMA
- Dual-Ended DMA
- Arbitrated-Ended DMA
- Interleaved DMA

- **Single-Ended DMA:** Single-Ended DMA Controllers operate by reading and writing from a single memory address. They are the simplest DMA.
- **Dual-Ended DMA:** Dual-Ended DMA controllers can read and write from two memory addresses. Dual-ended DMA is more advanced than single-ended DMA.
- **Arbitrated-Ended DMA:** Arbitrated-Ended DMA works by reading and writing to several memory addresses. It is more advanced than Dual-Ended DMA.
- **Interleaved DMA:** Interleaved DMA are those DMA that read from one memory address and write from another memory address.



Working Diagram of DMA Controller

Advantages of DMA Controller

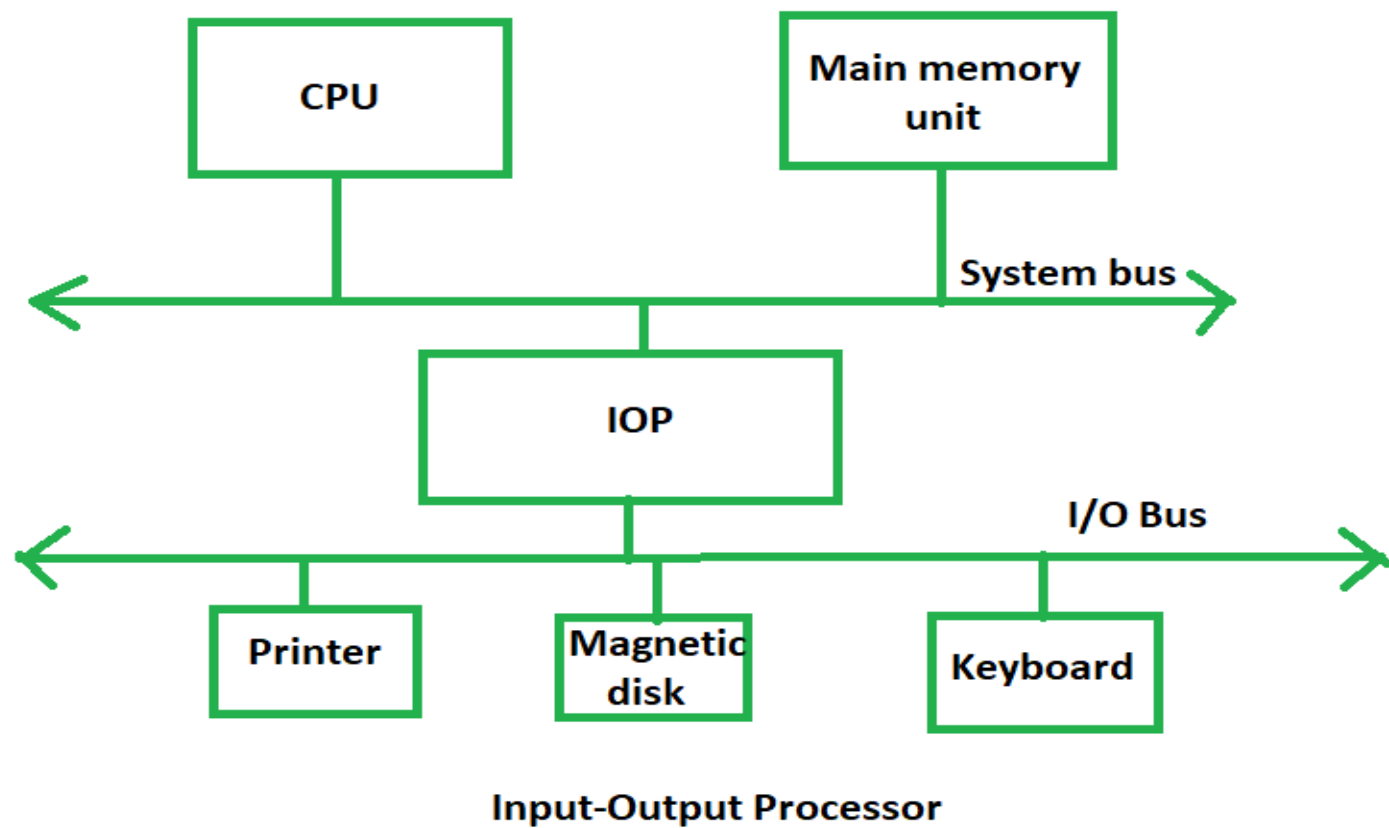
- Data Memory Access speeds up memory operations and data transfer
- CPU is not involved while transferring data.
- DMA requires very few clock cycles while transferring data.
- DMA distributes workload very appropriately.
- DMA helps the CPU in decreasing its load.

Disadvantages of DMA Controller

- Direct Memory Access is a costly operation because of additional operations.
- DMA suffers from Cache-Coherence Problems.
- DMA Controller increases the overall cost of the system.
- DMA Controller increases the complexity of the software.

I/O Processor

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



Advantages of Input-Output Processor

- The I/O devices can directly access the main memory without the intervention of the processor in I/O processor-based systems.
- It is used to address the problems that arise in the Direct memory access method.
- Reduced Processor Workload
- Improved Data Transfer Rates
- Increased System Reliability
- Scalability , Flexibility

Disadvantages of Input-Output Processor

- **Cost:** I/O processors can add significant costs to a system due to the additional hardware and complexity required. This can be a barrier to adoption, especially for smaller systems.
- Increased Complexity
- Limited Performance Gains
- Synchronization Issues
- Lack of Standardization



Thanks!

Assistant Professor
Shashank Sharma

Do you have any questions?
Shashank.16596@gmail.com
+91-9827881405