UNIT 5 (COA)

Peripheral devices

Peripheral devices are external devices connected to a computer, extending its capabilities. These devices don't typically perform the core functions of a computer but complement its operation, facilitating input, output, and storage.

Common peripheral devices include:

1. **Input Devices:** These devices input data into the computer. Examples include keyboards, mice, scanners, webcams, and microphones.
2. **Output Devices:** They display or output information from the computer. Monitors, printers, projectors, and speakers fall into this category.
3. **Storage Devices:** Used for storing data externally. Examples are hard drives, solid-state drives (SSDs), USB flash drives, and memory cards.
4. **Networking Devices:** These allow computers to connect and communicate with each other, like routers, modems, and network cards.
5. **Peripheral Component Interconnect (PCI) Cards:** Expansion cards installed inside the computer to enhance its capabilities, such as graphics cards, sound cards, and network interface cards (NICs).

Each peripheral serves a specific function, enhancing the overall functionality of the computer system.

I/O Interface

Input/output (I/O) interfaces are crucial components of any system that enable communication between different devices or components. These interfaces facilitate the transfer of data or signals between the CPU (Central Processing Unit) and external devices like keyboards, monitors, printers and more. They ensure that devices can interact with the computer system effectively.

I/O interfaces can take various forms, such as:

1. **Ports:** Physical connectors on a device that allow for the connection of cables or other hardware. Examples include USB ports, HDMI ports, Ethernet ports, and audio jacks.
2. **Buses:** These are pathways or connections that allow data to travel between components within a computer system. Buses can be internal (connecting components inside a computer) or external (connecting peripheral devices to a computer).
3. **Controllers:** Hardware components responsible for managing communication between the CPU and peripheral devices. Examples include disk controllers, network controllers, and USB controllers.
4. **Drivers:** Software programs that enable the operating system to communicate with hardware devices. They act as intermediaries, translating commands from the operating system into a language that the hardware understands and vice versa.

The efficiency and speed of data transfer between devices often rely on the type and capability of the I/O interface used. Different interfaces have varying data transfer rates, protocols, and compatibility with different devices.

Modern computing systems have evolved to incorporate faster and more versatile I/O interfaces to accommodate the growing demands of users for faster data transfer, higher resolutions, and increased device connectivity.

Interrupt

An interrupt in computing refers to a signal sent to the CPU (Central Processing Unit) by hardware devices, software, or external sources to temporarily halt the current program or process and divert the CPU's attention to another task that requires immediate processing.

When an interrupt occurs, the CPU suspends its current execution and transfers control to a specific interrupt handler routine. This routine performs actions associated with the interrupt, such as servicing a hardware request, handling an error condition, responding to user inputs, or executing a specific function requested by software.

Interrupts in computing can be categorized into various types based on their origin, priority, and purpose. Here are some common types:

1. **Hardware Interrupts:** Signals generated by hardware devices to request attention from the CPU. They can be further classified into:

    - **Maskable Interrupts:** Can be delayed or ignored by the CPU based on its current state and priority. In maskable interrupts, response time is high. The term "RST6.5 interrupt" typically refers to a specific type of interrupt on Intel 8085 microprocessors is example of Maskable Interrupt.

    - **Non-maskable Interrupts (NMI):** Can't be ignored or delayed; they require immediate attention by the CPU. In non-maskable interrupts, response time is low. Trap of 8085 microprocessor is an example for non-maskable interrupt.

2. **Software Interrupts:** Initiated by programs or software to perform specific tasks. These can include system calls or exceptions generated by programs.

3. **Internal Interrupts:** the terms "trap," "fault," and "abort" refer to different types of events or conditions that can occur during program execution

    - **Trap:** A software-generated interrupt caused by an error or an exception condition. They require special handling by OS when occured, such as dividing by zero.

    - **Fault:** A fault, also known as a page fault occurs when a program attempts to access a memory location that is not currently in physical memory (page fault). Faults are unintentional errors that can be addressed by the operating system. In the case of a page fault, the operating system may bring the required memory page into physical memory from secondary storage

    - **Abort:** Indicates a severe error or condition where the program cannot be restarted. An abort is a more severe and often unrecoverable event that results in the termination of a program or process. Aborts can occur due to critical errors, violation of system policies, or other exceptional conditions that cannot be safely handled.

In summary, traps are intentional software interrupts used for system calls and handling specific events, faults are unintentional errors related to memory access, and aborts are severe and often unrecoverable events that lead to the termination of a program or process

4. **External Interrupts:** Triggered by external devices connected to the computer system, such as keyboard input, mouse movements, or other peripherals.
5. **Vectored Interrupt** : In a vectored interrupt system, when an interrupt occurs, the interrupting device provides additional information (a vector / an identification code) to the processor about the type or source of the interrupt.
6. **Non-Vectored Interrupt**: In a non-vectored interrupt system, the interrupting device does not provide specific information about the interrupt source. Instead, the processor needs to poll or check each interrupt source to identify which one needs attention.

Exception

An exception, in computing, refers to an anomalous or exceptional condition that occurs during the execution of a program or a process, deviating from its normal flow. Exceptions are often caused by errors, unexpected situations, or specific events that disrupt the regular execution of code.

Exceptions can occur due to various reasons, including:

1. **Runtime Errors:** Occur during program execution due to invalid operations, such as dividing by zero, accessing invalid memory, or trying to perform unsupported operations.
2. **Input/Output Errors:** Arise when there's an issue with reading or writing data to and from external sources, like files or devices.
3. **Resource Unavailability:** Happens when a requested resource, like memory or a file, is not accessible or unavailable.
4. **Logic Errors:** When a program encounters a situation that doesn't align with the expected behavior, but no explicit error is thrown.

Modes of Data Transfer

In computer organization and architecture (COA), there are several modes of transfer, but three of the most common are:

1. Programmed I/O transfer

2. Interrupt-driven or Interrupt Initiated I/O transfer

3. Direct Memory Access (DMA)

1. Programmed I/O transfer

Programmed I/O transfer is a data transfer mode in computer organization and architecture. In this mode of transfer***, data is transferred between an I/O device and the computer's memory under the control of a program***. The central processing unit (CPU) actively manages the transfer, sending commands to the I/O device to initiate the transfer and monitor its progress.

In a programmed I/O transfer, the CPU sends a command to the I/O device to initiate the transfer and then waits for the transfer to complete. During the transfer, the CPU cannot perform other tasks as it is busy managing the transfer.

Programmed I/O transfer provides low-level control over the transfer, as the CPU can issue commands to the I/O device to control the transfer. This can be useful for certain applications where fine-grained control over the transfer is required. However, it also means that the CPU is tied up during the transfer, potentially slowing down other tasks that the computer performs.

Programmed I/O transfer is typically slower than other transfer modes, such as direct memory access (DMA). However, it can still be useful in certain situations, such as when low-level control over the transfer is required or when the amount of data is small.

Programmed I/O transfer works as follows:

1. The CPU sends a command to the I/O device to initiate the transfer. This command may specify the transfer direction (i.e., from memory to the I/O device or from the I/O device to memory), the starting address in memory where the data is to be stored or retrieved, and the number of bytes to be transferred.

2. The I/O device receives the command and begins the transfer. It transfers the data to or from the specified memory location, one byte at a time.

3. The CPU repeatedly polls the status register of the I/O device to monitor the progress of the transfer. The status register contains information about the current state of the I/O device, including whether the transfer is complete.

4. When the transfer is complete, the I/O device sets a flag in the status register to indicate that the transfer is done. The CPU reads the status register, and when it sees the flag, it knows that the transfer is complete.

5. The CPU can resume other tasks after the transfer is finished. In programmed I/O transfer, the CPU manages the transfer and monitors its progress.

2. Interrupt-driven Transfer

The interrupt-driven transfer is a data transfer mode in computer organization and architecture. In this transfer mode, ***the CPU is notified of incoming data through an interrupt request.*** An interrupt is a signal sent to the CPU indicating that an I/O device has data ready for transfer.

When the CPU receives an interrupt, it suspends its current task and handles it. The CPU communicates with the I/O device to initiate the data transfer and then waits for the transfer to complete. Once the transfer is complete, the CPU returns to its previous task.

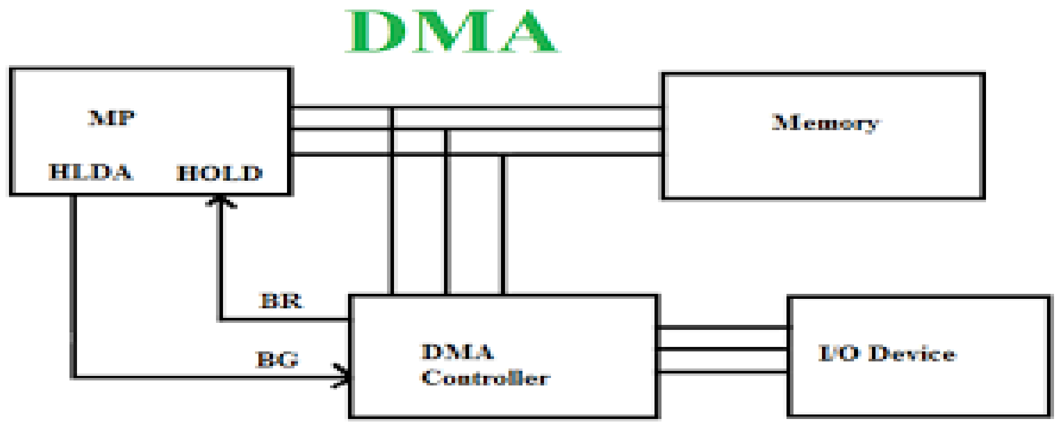
The interrupt-driven transfer balances the control offered by programmed I/O transfer and the speed offered by direct memory access (DMA). The CPU controls the transfer, but the transfer occurs in the background, freeing up the CPU to perform other tasks. This transfer mode is commonly used in systems requiring high-speed data transfer and low-level control over the transfer. Examples include keyboard input, serial communication, disk I/O, and display output.

3. Direct Memory Access (DMA)

Direct Memory Access (DMA) is a data transfer method in computer organization and architecture that allows an I/O device to transfer data directly to or from memory without the involvement of the CPU. DMA allows I/O devices to transfer data to or from memory at high speeds without putting additional strain on the CPU.

In DMA, a DMA controller is used to manage data transfer between the I/O device and memory. The DMA controller is responsible for setting up the transfer, starting it, and monitoring it to ensure it is completed successfully. The CPU is not involved in the actual data transfer but is notified when it is complete.

DMA is often used for high-bandwidth I/O devices, such as disk drives, network interfaces, and graphics cards, where the CPU is not fast enough to manage the data transfer in real time. DMA allows these devices to transfer data to or from memory at high speeds without putting additional strain on the CPU.



Direct Memory Access (DMA) works as follows:

1. Initialization: The CPU initializes the DMA transfer by setting up the source and destination addresses, the size of the transfer, and any other relevant parameters.

2. Request: The I/O device requests a DMA transfer by asserting a request signal to the DMA controller.

3. Grant: The DMA controller grants the request and starts the transfer by asserting a grant signal to the I/O device.

4. Data Transfer: The I/O device transfers the data directly to or from memory without involving the CPU. The DMA controller manages the transfer and provides necessary addresses and control signals.

5. Completion: When the transfer is complete, the I/O device asserts a completion signal to the DMA controller. The DMA controller then informs the CPU that the transfer is complete.

The CPU can perform other tasks during the transfer as the DMA controller handles the data transfer. This allows the CPU to improve its overall performance and responsiveness.

Overall, DMA provides a high-speed data transfer method that allows I/O devices to transfer data to or from memory without putting additional strain on the CPU. This improves system performance and responsiveness, especially for high-bandwidth I/O devices.

There are three different modes of DMA data transfer which are as follows

- **Burst Mode** − In burst mode, a whole block of data is shared in one contiguous sequence. Since the DMA controller is allowed access to the system buses by the CPU, it sends all bytes of data in the data block earlier yield control of the system buses back to the CPU. This mode is beneficial for loading programs or data records into memory, but it does provide the CPU inactive for associatively long periods.

- **Cycle Stealing mode** − In cycle stealing mode, the DMA controller gets access to the system buses as in burst mode, using the BR and BG signals. It can share one byte of information and then deasserts BR, returning control of the system buses to the CPU. It already issues requests via BR, sharing one byte of information per request, just before it has shared its whole block of data.

- **Transparent Mode** − Transparent mode needed the most time to share a block of data, yet it is also important in terms of whole system performance. In transparent mode, the DMA controller only shares data when the CPU is implementing operations that do not use the system buses. For example, the relatively simple CPU has multiple states that change or process data only within the CPU.

I/O channels and processors

I/O (Input/Output) channels and processors play crucial roles in computer systems, facilitating communication between the CPU, memory, and external devices. Here's an overview of each:

1. **I/O Channels:**

    - **Definition:** I/O channels are pathways or communication channels that allow data to be transferred between the CPU and external devices.
    - **Purpose:** They serve as interfaces for data transfer between the central processing unit (CPU) and peripherals like storage devices, network interfaces, display units, and input devices.
    - **Characteristics:** I/O channels are designed to handle various types of data transfers, including data input (from peripherals to the CPU) and data output (from the CPU to peripherals).

2. **I/O Processors:**

    - **Definition:** I/O processors, or I/O controllers, are specialized processors dedicated to managing the input and output operations between the CPU and external devices.
    - **Purpose:** They offload the CPU by handling the details of data transfer, error checking, and other low-level operations associated with I/O operations. This allows the CPU to focus on computation rather than managing individual devices.
    - **Characteristics:** I/O processors typically have their own memory, control logic, and communication interfaces. They can operate concurrently with the CPU, managing multiple I/O operations simultaneously.

In summary, I/O channels represent the communication pathways between the CPU and external devices, while I/O processors are specialized units that assist in managing and controlling the flow of data between the CPU and these devices. The use of I/O channels and processors enhances the overall efficiency of a computer system by providing dedicated resources for handling I/O operations, freeing up the CPU to perform other computational tasks.

Serial Communication

Serial communication is a method of transferring data between two or more devices by sending the data as a stream of bits over a single wire or communication channel. In serial communication, data is transmitted sequentially, one bit at a time, over a single communication path. This is in contrast to parallel communication, where multiple bits are sent simultaneously over separate channels.

There are two main types of serial communication:

1. **Asynchronous Serial Communication:**
    - **Characteristics:**
        - Each byte of data is framed by start and stop bits.
        - The timing is not synchronized between the sender and receiver.
        - It is more flexible and tolerant of variations in timing.
    - **Examples:**
        - RS-232 (Recommended Standard 232) is a common asynchronous serial communication standard used for connecting devices such as computers and modems.
        - Asynchronous Transfer Mode (ATM) is a telecommunications standard that can use asynchronous serial communication.
2. **Synchronous Serial Communication:**
    - **Characteristics:**
        - Data is transmitted in a continuous stream without start and stop bits.
        - Timing is synchronized using a clock signal shared between the sender and receiver.
        - It is more susceptible to timing issues if devices are not well-synchronized.
    - **Examples:**
        - I2C (Inter-Integrated Circuit) is a synchronous serial communication protocol often used for communication between integrated circuits.

- SPI (Serial Peripheral Interface) is another synchronous serial communication protocol commonly used for short-distance communication between microcontrollers and peripheral devices.

Common Elements in Serial Communication:

1. **Start and Stop Bits:** In asynchronous communication, each byte is framed by start and stop bits, providing a mechanism for synchronization.
2. **Baud Rate:** Baud rate refers to the speed at which bits are transmitted per second. It is a crucial parameter in serial communication, determining the rate of data transfer.
3. **Data Bits and Parity:** The number of data bits per frame and whether parity checking is used are important settings in serial communication configurations.

Serial communication is widely used in various applications, including connecting peripheral devices to computers, communication between microcontrollers, and interfacing with sensors and other embedded systems. It provides a simple and efficient way to transfer data over relatively long distances with minimal wiring.

Difference between Synchronous and asynchronous communication

S. No.	**Synchronous Communication**	**Asynchronous Communication**
1.	In Synchronous transmission, data is sent in form of blocks or frames.	In Asynchronous transmission, data is sent in form of bytes or characters.
2.	Synchronous transmission is fast.	Asynchronous transmission is slow.
3.	Synchronous transmission is costly.	Asynchronous transmission is economical.
4.	In Synchronous transmission, the time interval of transmission is constant.	In Asynchronous transmission, the time interval of transmission is not constant, it is random.
5.	In this transmission, users have to wait till the transmission is complete before getting a response back from the server.	Here, users do not have to wait for the completion of transmission in order to get a response from the server.
6.	The start and stop bits are not used in transmitting data.	The start and stop bits are used in transmitting data that imposes extra overhead.
7.	Synchronous transmission needs precisely synchronized clocks for the information of new bytes.	Asynchronous transmission does not need synchronized clocks as parity bit is used in this transmission for information of new bytes.
8.	Errors are detected and corrected in real time.	Errors are detected and corrected when the data is received.
9.	Low latency due to real-time communication.	High latency due to processing time and waiting for data to become available.
10.	Examples: Telephonic conversations, Video conferencing, Online gaming.	Examples: Email, File transfer,Online forms.

Standard Communication Interface

Standard communication interfaces provide a set of specifications and protocols that allow different devices and systems to communicate with each other. These standards ensure compatibility and interoperability, enabling devices from different manufacturers to work together seamlessly. Here are some common standard communication interfaces:

1. **USB (Universal Serial Bus):**
   - **Description:** USB is a widely used standard for connecting devices to computers and other hosts. It supports data transfer, power supply, and various device classes like keyboards, mice, printers, and storage devices.
   - **Versions:** USB 1.0, USB 2.0, USB 3.0, USB 3.1, USB 3.2, and USB4.
2. **Ethernet:**
   - **Description:** Ethernet is a standard for wired local area networking (LAN). It defines how data packets are placed on the network and transmitted between devices. It is commonly used for internet connectivity and intranet communication.
   - **Versions:** Ethernet, Fast Ethernet, Gigabit Ethernet, 10 Gigabit Ethernet, and more.
3. **Bluetooth:**
   - **Description:** Bluetooth is a wireless communication standard for short-range connections between devices. It is often used for connecting peripherals like keyboards, mice, headphones, and for data exchange between smartphones and other devices.
   - **Versions:** Bluetooth 1.x, 2.x, 3.x, 4.x, 5.x.
4. **Wi-Fi (IEEE 802.11):**
   - **Description:** Wi-Fi is a set of wireless communication standards for local area networking. It allows devices to connect to a network wirelessly, providing internet access and data transfer.
   - **Versions:** 802.11a, 802.11b, 802.11g, 802.11n, 802.11ac, 802.11ax.
5. **HDMI (High-Definition Multimedia Interface):**
   - **Description:** HDMI is a standard for transmitting high-definition audio and video signals between devices like TVs, monitors, and audio-visual equipment.
   - **Versions:** HDMI 1.x, HDMI 2.x.
6. **PCI Express (PCIe):**
   - **Description:** PCIe is a high-speed interface standard used for connecting various internal components within a computer, such as graphics cards, storage devices, and network cards.
   - **Versions:** PCIe 1.0, PCIe 2.0, PCIe 3.0, PCIe 4.0.

7. **Serial ATA (SATA):**
    - **Description:** SATA is a standard for connecting storage devices such as hard drives and solid-state drives to a computer's motherboard.
    - **Versions:** SATA 1.0, SATA 2.0, SATA 3.0.
8. **SPI (Serial Peripheral Interface):**
    - **Description:** SPI is a synchronous serial communication interface commonly used for short-distance communication between microcontrollers and peripheral devices.
    - **Characteristics:** Master-slave architecture, multiple slave devices can be connected to a single master.

These standards play a crucial role in ensuring seamless connectivity and communication between diverse devices in modern computing and communication ecosystems. They are established by industry organizations and are often adopted globally to facilitate widespread compatibility.