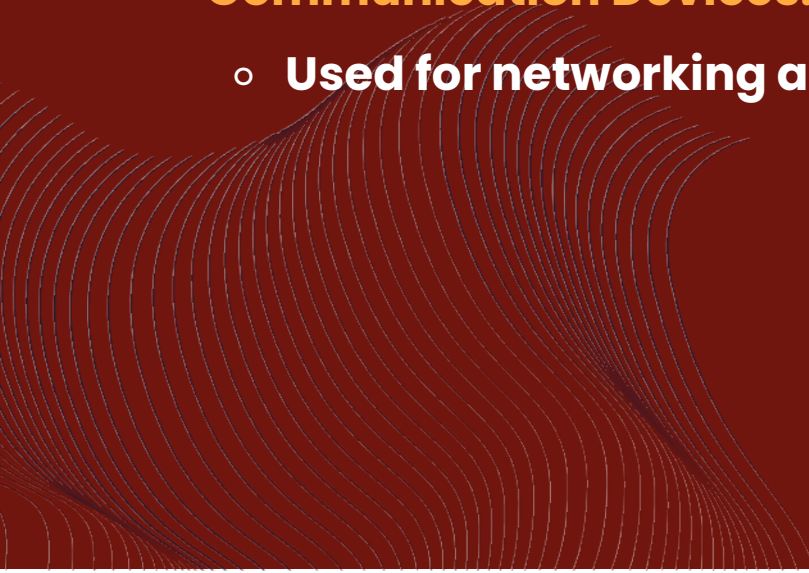


# Input/Output (I/o)

---

# Categories of I/O Devices

- **Human Readable:**
    - Devices that interact with users (e.g., monitors, printers).
  - **Machine Readable:**
    - Devices for electronic communication (e.g., sensors, robots).
  - **Communication Devices:**
    - Used for networking and remote communication (e.g., modems, routers).
- 
- A decorative graphic consisting of numerous thin, white, wavy lines that flow from the left side of the slide towards the center, creating a sense of motion and depth.

# Key Differences Across I/O Devices

- **Data Rate:**
  - Devices vary significantly in how fast they transfer data.
- **Application:**
  - The use of a device affects its integration into the operating system and utilities.
- **Complexity of Control:**
  - Some devices, like printers, are simple to control, while others, like disks, are more complex.
- **Unit of Transfer:**
  - Devices may transfer data as streams of bytes or as larger blocks.

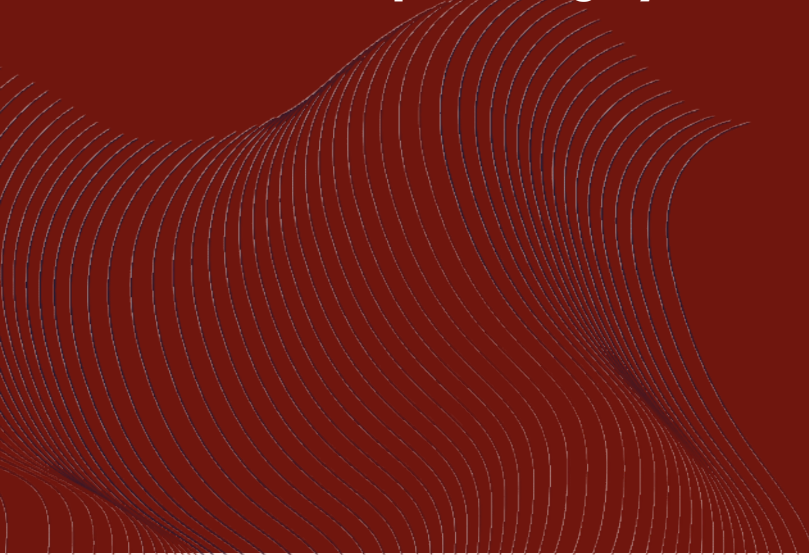
---

# Additional Differences in I/O Devices

- **Data Representation:**

- **Devices use different encoding schemes, affecting how data is handled.**

- **Error Conditions:**

- **Error handling and reporting methods vary greatly, adding complexity for operating systems and user processes.**
- 
- A decorative graphic consisting of numerous thin, white, wavy lines that flow from the left side of the slide towards the center, creating a sense of motion and depth against the dark red background.

# Approaches to I/O Operations

- **I/O Function Methods:**

- **Programmed I/O:** Processor waits for I/O operations to complete.
- **Interrupt Driven I/O:** Processor executes other tasks while I/O completes, using interrupts to signal completion.
- **Direct Memory Access (DMA):** Transfers data directly between memory and devices without involving the processor for each transfer.

---

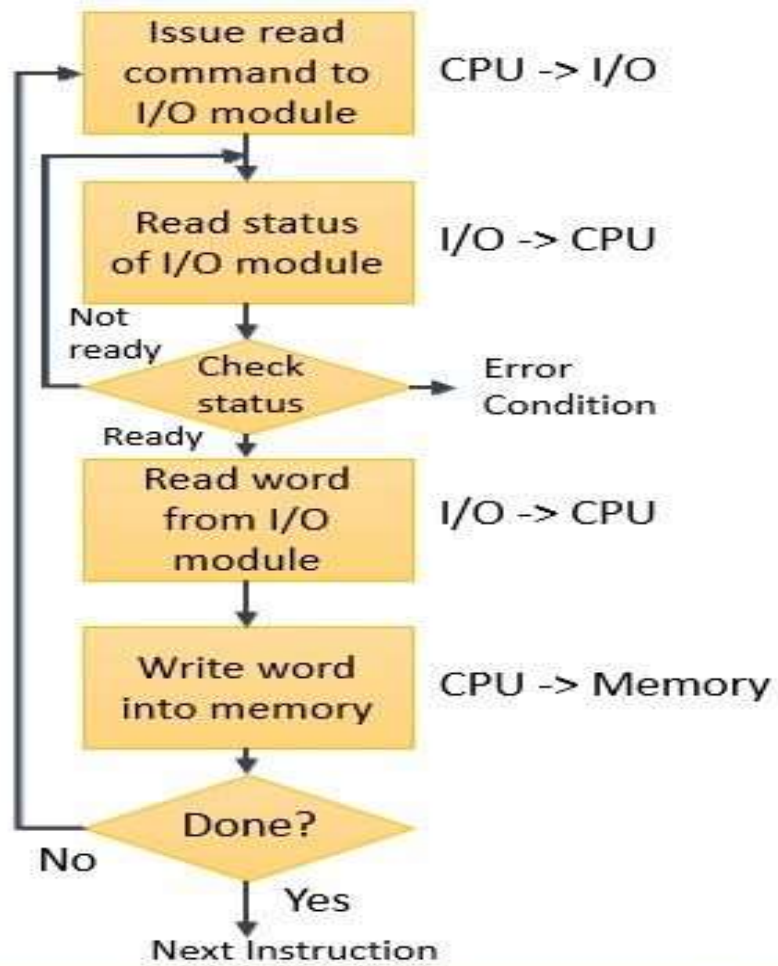
# Programmed I/O

**Programmed I/O (PIO) is a method of data transfer between the CPU and an I/O device where the CPU directly controls all aspects of the data transfer. In this method, the CPU actively monitors the status of the I/O device and**

A decorative graphic consisting of numerous thin, white, wavy lines that flow from the left side of the slide towards the center, creating a sense of movement and depth.

Here's a breakdown of how it works:

1. Polling (Active Waiting): The CPU sends a command to the I/O device (such as read or write) and then continuously checks or polls the device's status register to see if it is ready to receive or send data.
2. Busy-Waiting: While the CPU is polling, it is busy waiting, meaning it can't perform any other operations until the I/O device signals that it is ready. This leads to inefficient use of CPU time, as the CPU is blocked from doing other tasks during this period.
3. Data Transfer: Once the I/O device is ready, the CPU transfers data between the device and memory.
4. Completion: After the I/O operation is complete, the CPU continues with the next task.



Programmed I/O to transfer data  
from I/O module to memory



## Advantages:

- **Simple to Implement:** Programmed I/O is straightforward to implement and requires minimal hardware.
- **Suitable for Simple Devices:** It works well for slow or simple devices where the time spent polling is not a major concern (e.g., a keyboard).

## Disadvantages:

- **Inefficient Use of CPU:** The CPU is occupied waiting for the I/O device, which reduces overall system performance, especially with slow I/O devices.
- **Limited Scalability:** In systems with many I/O devices, programmed I/O becomes inefficient, as the CPU must spend a significant amount of time managing I/O.

## **Interrupt Driven I/O**

Interrupt-Driven I/O is a method of data transfer between the CPU and I/O devices that allows the CPU to perform other tasks while waiting for I/O operations to complete. Instead of the CPU actively polling the I/O device for its status, the device generates an interrupt signal when it requires attention, enabling more efficient use of CPU resources.

# How Interrupt-Driven I/O Works:

**I/O Request:** The CPU sends a command to the I/O device (e.g., read or write operation).

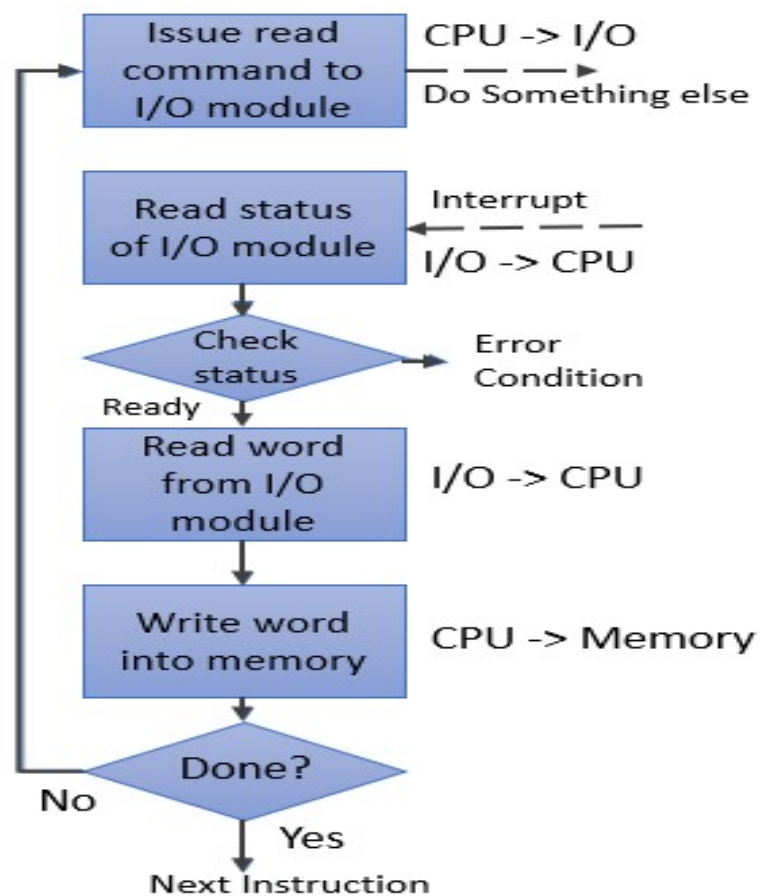
**Background Processing:** After issuing the command, the CPU continues executing other instructions without waiting for the I/O operation to finish.

**Interrupt Generation:** When the I/O device completes the requested operation (e.g., data is ready to be read), it sends an interrupt signal to the CPU.

**Interrupt Handling:** The CPU pauses its current execution, saves its state (context switching), and jumps to a predefined interrupt service routine (ISR) that addresses the I/O operation.

**Data Transfer:** The CPU or the ISR processes the data transfer (e.g., reading data from the device into memory).

**Resume Operation:** Once the I/O operation is complete and the interrupt has been handled, the CPU restores its previous state and continues executing the interrupted task.



**Interrupted I/O to Transfer Data  
from I/O Module to Memory**

## Advantages:

**Efficient CPU Utilization:** The CPU can perform other computations while waiting for I/O operations, leading to better overall system performance.

**Responsive Systems:** Systems can respond more quickly to I/O events without being blocked by polling.

**Simplified I/O Management:** The interrupt mechanism handles many complexities associated with I/O operations.

## Disadvantages:

**Overhead of Interrupt Handling:** Each interrupt requires context switching, which can introduce overhead, especially if interrupts occur frequently.

# Direct Memory Access (DMA)

- **DMA Overview:**

- **Manages data transfer between memory and I/O devices without constant processor involvement.**
- **Reduces the need for processor intervention, only interrupting the processor after the entire data block is transferred.**

---

# DMA Process

- **How DMA Works:**
  - **Processor sends commands to the DMA module, specifying:**
    - **Read or write operation.**
    - **Device address.**
    - **Memory address.**
    - **Data block size.**
- **DMA module handles the data transfer autonomously.**

## Direct Memory Access (DMA)

is an efficient method of data transfer between an I/O device and the main memory that allows certain hardware subsystems to access the main system memory independently of the CPU.

This approach enhances performance by offloading data transfer tasks from the CPU, enabling it to execute other instructions while the data transfer is being handled



## How DMA Works:

**DMA Controller:** A dedicated hardware component, known as the DMA controller, manages the data transfer between the I/O device and memory.

**CPU Initialization:** The CPU initiates a DMA transfer by sending a request to the DMA controller. This request typically includes:

- The address in memory where data will be transferred.

- The address of the I/O device.

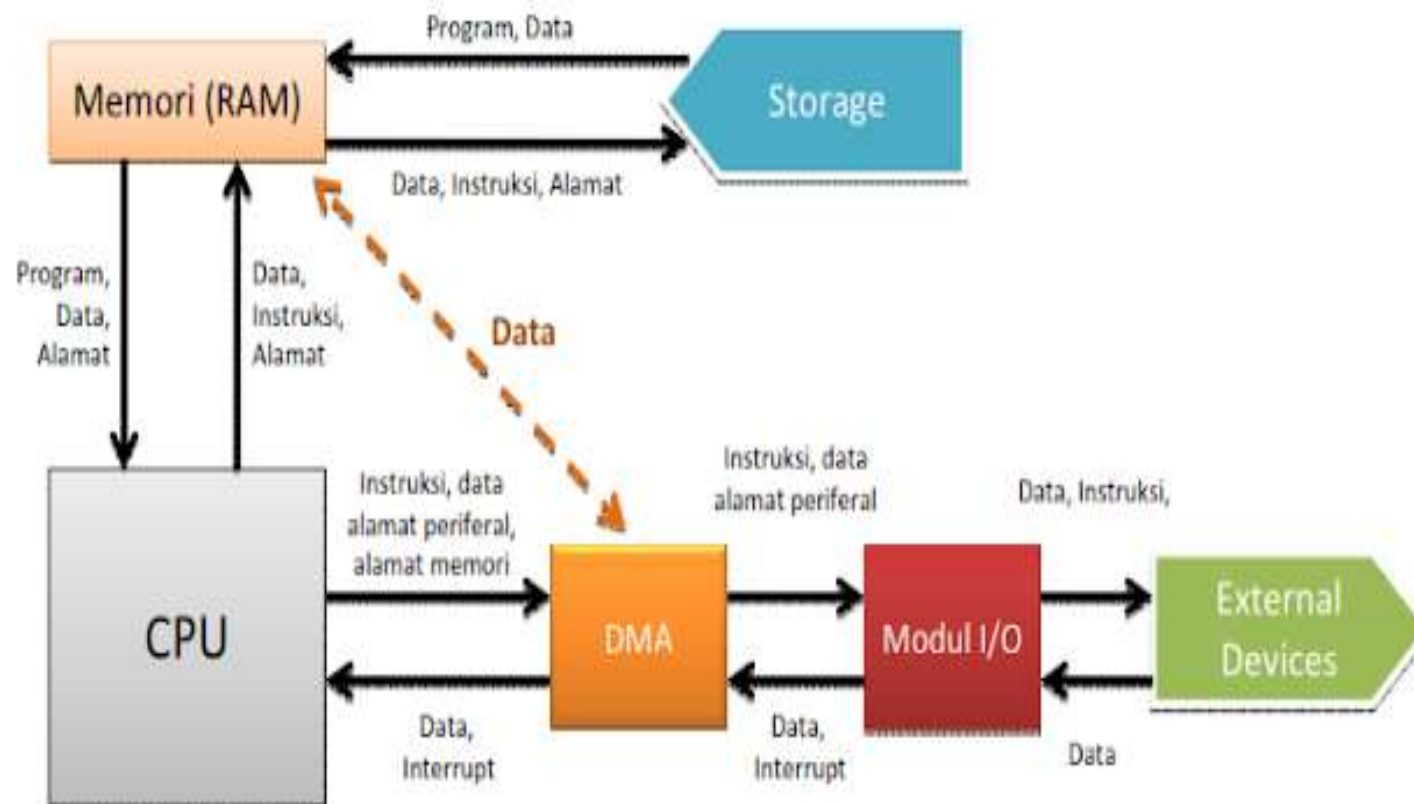
- The amount of data to be transferred.

**Bus Control:** Once the DMA controller is configured, it takes control of the system bus, allowing it to read data directly from the I/O device and write it to memory without CPU intervention.

**Data Transfer:** The DMA controller performs the data transfer, managing the read and write operations until the specified amount of data has been moved.

**Completion Notification:** After the data transfer is complete, the DMA controller sends an interrupt to the CPU to notify it that the operation has finished, allowing the CPU to resume processing other tasks.

# Direct Memory Access



---



Do you have any questions?



**THANK YOU!**