UNIT- 4 Input Output Organisation

Input Output Organization

- Welcome to the world of Input-Output Organization! This chapter is all about how your computer communicates with the outside world, especially through devices like keyboards, mice, and printers—these are known as peripheral devices. Think of them as the helpers that make your computer do things. Ever wondered how your computer understands when you click your mouse or type on your keyboard? That's exactly what we're going to explore. We'll break down the basics of Input-Output Organization, demystifying the process of how your computer interacts with these peripheral devices. So, let's dive in and uncover the secrets behind the scenes of how your digital world connects with the physical through these devices!

Peripheral Devices

These are nothing but input/output device which are connected to the computer. These are of 3 types

1. Input Peripheral
2. Output Peripheral
3. Input-output peripheral

Input Peripheral

Input peripherals play a crucial role in providing the computer with the necessary data to process. These devices serve as the bridge between human interaction and the digital realm, allowing users to input information effectively. Examples of input peripherals include keyboards, mice, and scanners.

- **Keyboard:** When we use a keyboard, we press keys, each of which corresponds to a meaningful character in the ASCII (American Standard Code for Information Interchange) set. ASCII uses 7 bits, enabling the representation of 128 characters. Among these, 94 are printable, and 34 are non-printable. Within the printable characters, 26 represent the alphabet, 10 represent digits (0-9), and 32 include upper and lower case letters along with special symbols.
- **Efficiency Considerations:** Recognizing that input devices like keyboards operate at a slower pace compared to the speed of the CPU, computers implement the concept of multiprogramming. This technique ensures that the CPU

remains busy and efficient, even when dealing with input devices that are inherently slower due to human operation.

Output Peripheral Devices

Output peripheral devices play a crucial role in presenting information to users in a comprehensible format. These devices are primarily designed to display data processed by the computer, offering a tangible output. Among the most commonly used output peripherals are:

- **Monitors:** Monitors serve as visual displays, presenting a wide range of information, from text and images to videos. They are the primary output interface for users to interact with the digital content produced by the computer.
- **Printers:** Printers, on the other hand, provide a physical representation of digital data. They convert electronic documents into tangible, printed copies, making information accessible beyond the digital realm.

Input-Output Peripheral Devices

Input-Output (I/O) peripheral devices are the versatile components that facilitate both the input and output of data between the computer and its external environment. These devices serve as bidirectional bridges, enabling communication between the user and the computer system. Examples of Input-Output peripheral devices include:

- **Touchscreen Displays:** Touchscreens are multifunctional devices that allow users to input data by touch and receive visual output simultaneously. They find applications in devices like smartphones, tablets, and interactive kiosks.
- **External Hard Drives:** External hard drives not only store data but also provide a means for users to input information into the computer (by transferring files to the computer) and receive output (by accessing stored data).

Input-Output Interface

- The Input-Output (I/O) interface serves as the crucial link for transferring data between the internal storage of a computer and external I/O devices. Establishing a communication link between peripherals and the CPU is essential for seamless data exchange.
- In most cases, the CPU cannot directly access I/O devices due to fundamental differences between the CPU and these devices. To overcome these disparities, an I/O interface is employed.

- By utilizing input and output interfaces, the CPU can effectively communicate with I/O devices, bridging the gap and facilitating the exchange of information.

Differences between CPU and IO devices

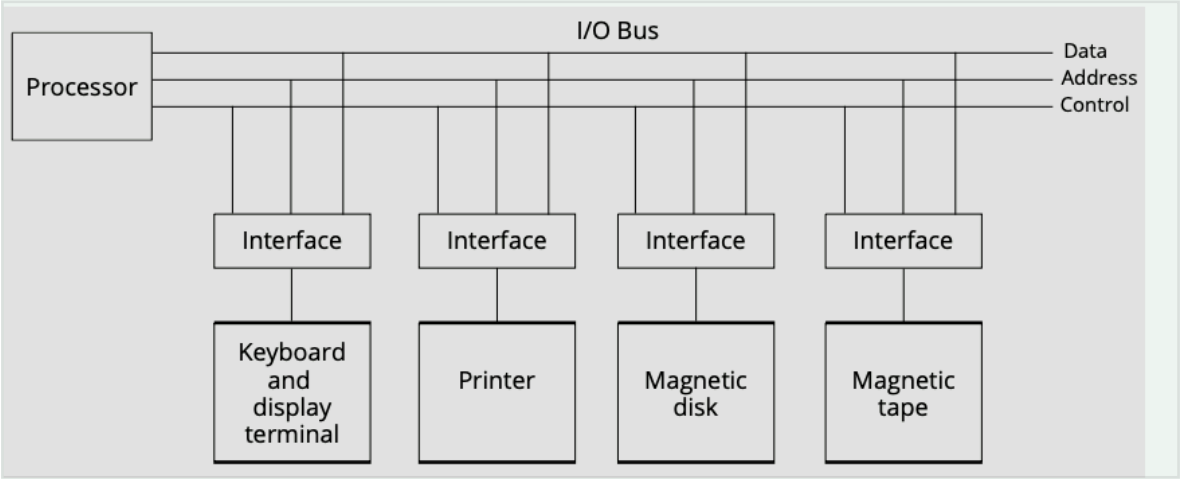
- Peripherals operate on electromechanical and electromagnetic principles, differing significantly from the functioning of the CPU and memory. Consequently, a signal conversion is necessary to facilitate communication.
- While CPU processing speed is considerably faster than that of IO devices, the latter are relatively slower. To synchronize these disparate speeds, an interface is employed to ensure smooth and efficient data transfer.
- The data format for IO devices is typically in bytes, whereas the CPU executes instructions in the form of words.
- Input-Output operations transfer data in a serial manner, whereas the CPU executes instructions in parallel, handling multiple instructions simultaneously.
- Each peripheral device has unique operational characteristics that must be controlled to prevent interference with the operations of other peripherals. Coordination and control are essential to ensure smooth system functionality.

To address and reconcile these differences, computer systems incorporate special hardware components known as interface units. These interface units serve as intermediaries between the CPU and peripherals, managing the communication and ensuring compatibility between the diverse elements of the computing system.

I/O Bus and Interface Modules

In the diagram below, you can see the communication link between the processor and peripherals. This link is facilitated by the I/O bus, which includes data lines, address lines, and control lines. Each peripheral device is equipped with its own interface unit, and these interfaces serve crucial functions:

- **Decoding:** The interface unit decodes the address and control signals received from the CPU, understanding the instructions for the peripheral.
- **Management:** It oversees and manages the operation of the peripheral device, ensuring its functions as intended.
- **Synchronization:** Using buffers, the interface unit synchronizes the speed of the CPU with that of the peripheral, facilitating smooth data transfer.



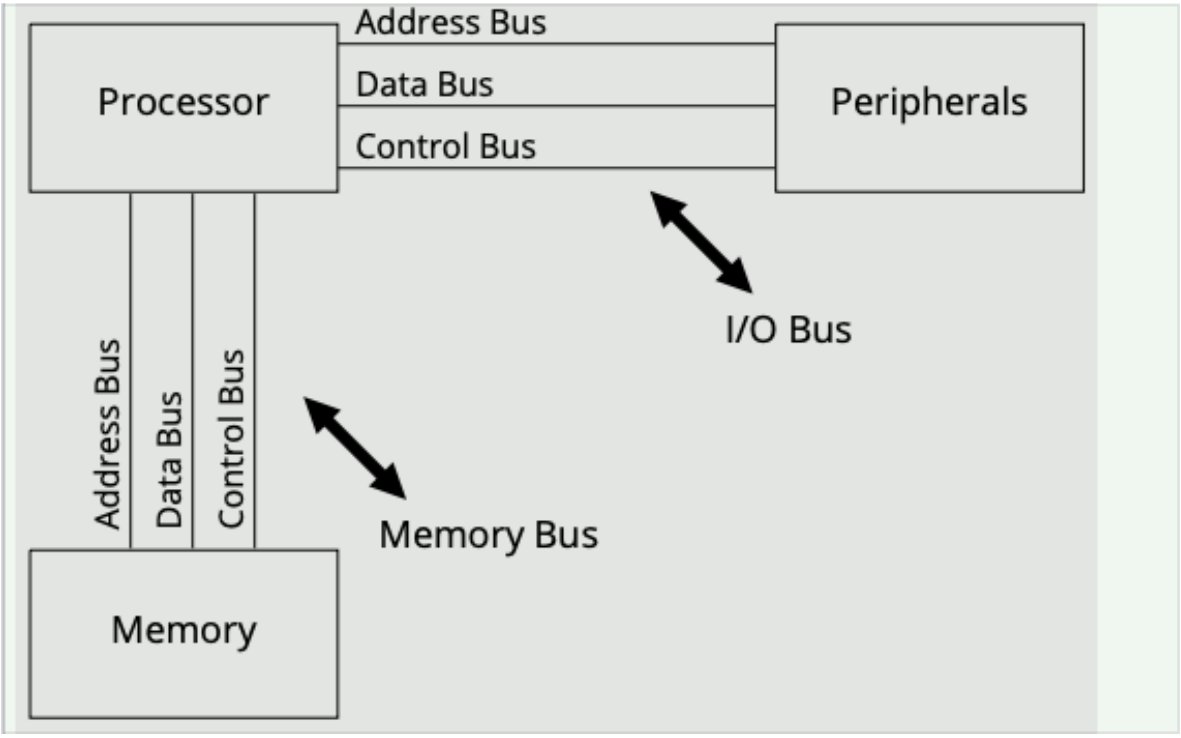
The I/O bus connects to all peripheral interfaces. To communicate with a specific device, the processor sends the device's address through the address lines. Each interface checks the incoming address against its own, activating the pathway between the bus lines and the designated device. If a peripheral's address doesn't match, its interface becomes inactive. The processor then sends control signals through the control lines and data through the data lines.

- **Processor Commands:** The processor can generate four types of commands for peripherals:
    1. **Control Command:** Instructs the device on what action to perform and how to execute it.
    2. **Status Command:** Checks and reports the current condition or status of the device.
    3. **Data Output Command:** Initiates the transfer of data from the device to the processor.
    4. **Data Input Command:** Prompts the device to take data from the data bus and store it in its buffer.

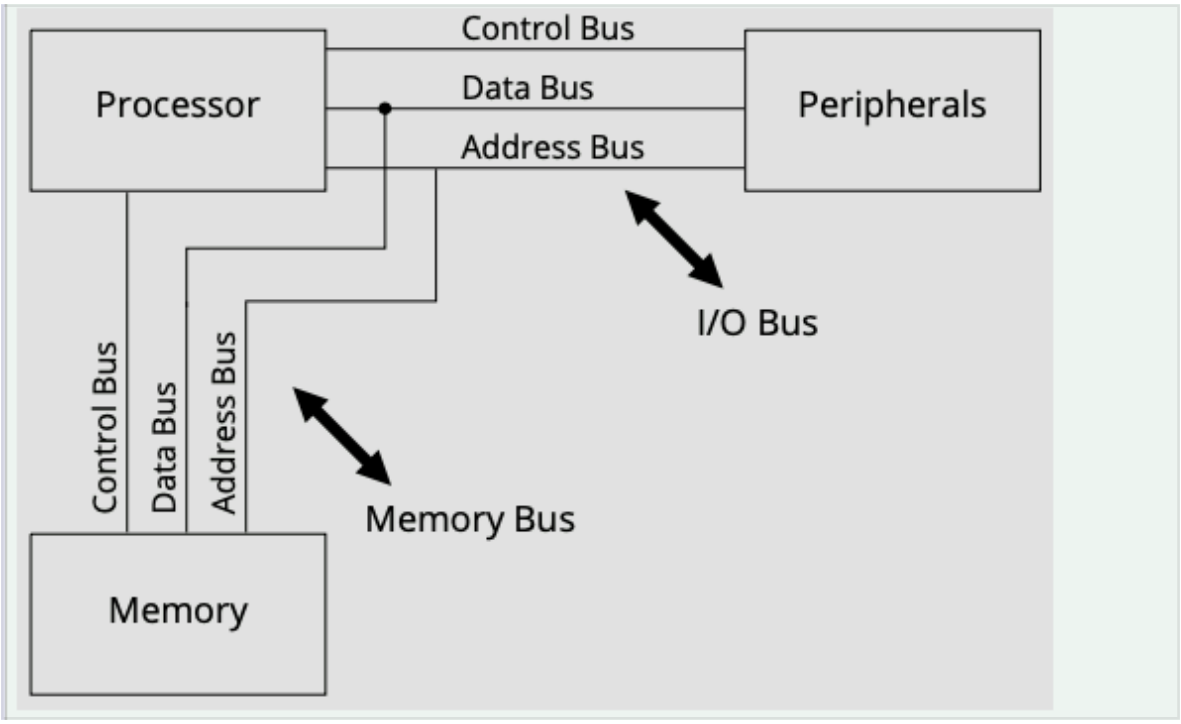
I/O Bus versus Memory Bus

The processor employs the I/O bus to communicate with peripherals, while the memory bus facilitates communication between the processor and memory. Both the I/O bus and memory bus share similar components, including data lines, address lines, and read/write control lines. The utilization of these buses can take different forms, and there are three main ways in which the I/O bus and memory bus can be configured:

1. Separate Buses: In this configuration, two distinct buses are employed—one dedicated to memory and the other to I/O operations. This approach ensures independent communication pathways, minimizing potential conflicts between memory and I/O processes. See the illustration below for a visual representation.

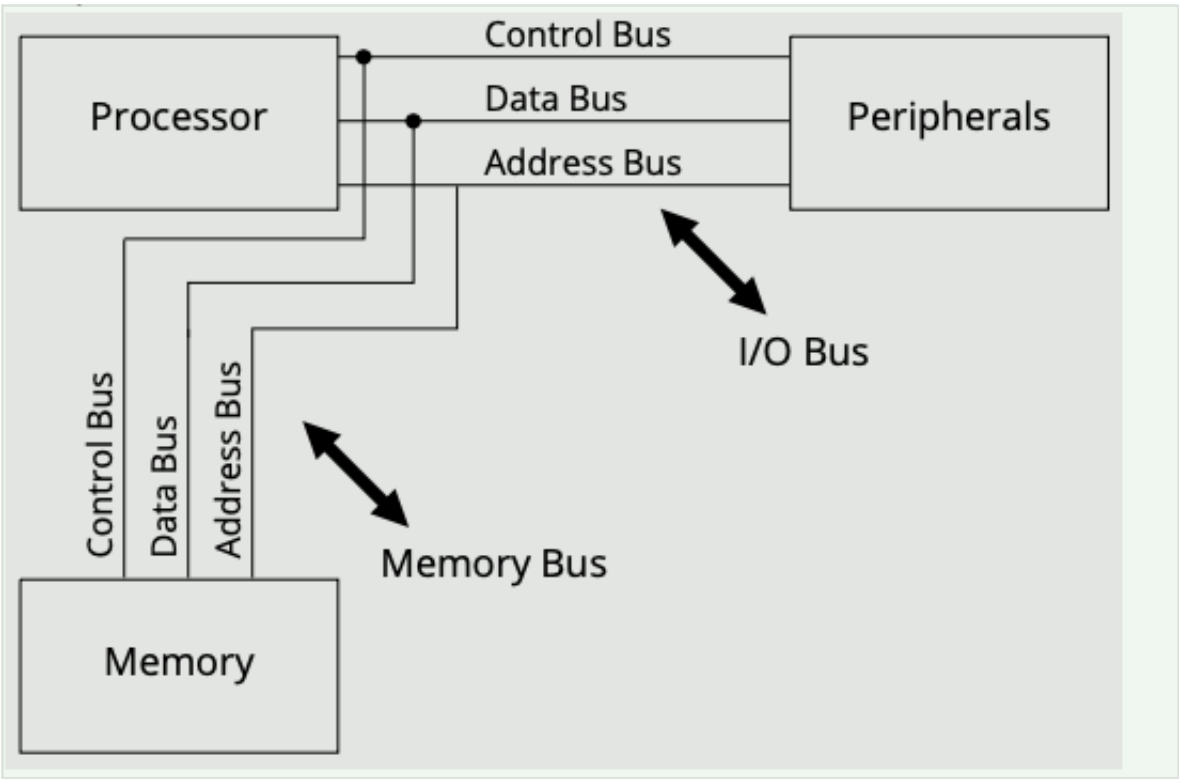


2. Common Bus with Separate Control Lines: An alternative configuration involves using a single bus for both memory and I/O operations. However, separate control lines are designated for each, allowing for independent management of memory and I/O processes. This approach optimizes resource sharing while maintaining control over the distinct functions. Refer to the diagram below for a clearer understanding.



3. Common Bus with Shared Control Lines: The third approach

integrates both memory and I/O operations onto a single bus, and the control lines are shared between the two. This configuration simplifies the overall design by reducing the number of dedicated control lines. The diagram below illustrates this consolidated setup.



Understanding the distinction between I/O and memory buses is crucial for designing efficient computer architectures. The choice of bus configuration depends on factors such as system complexity, performance requirements, and the need for resource optimization. By exploring these configurations, we gain insights into how computers manage data flow between the processor, memory, and peripherals, contributing to the overall functionality and performance of computing systems.

I/O Mapping A.K.A I/O Addressing A.K.A I/O Interfacing Technique

When the CPU shares a common bus system attached to both memory and I/O devices, it needs a way to differentiate between signals related to memory and those related to I/O. This is where Isolated I/O and Memory Mapped I/O come into play, providing distinct techniques for managing communication within the system.

- **Isolated I/O:** In this method, separate control lines are designated for memory and I/O operations. The CPU uses these control lines to efficiently manage the flow of information, ensuring that data reaches the intended destination without interference.

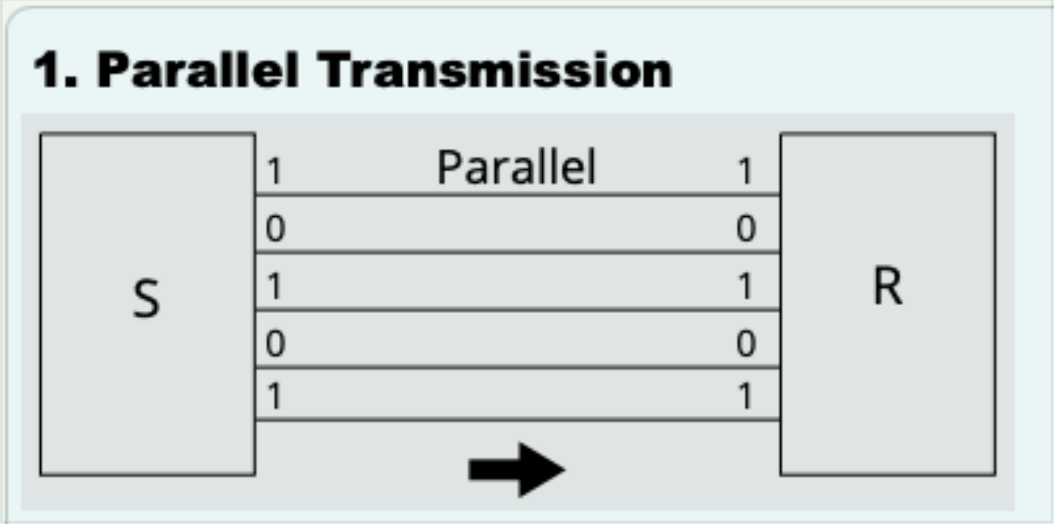
- **Memory Mapped I/O:** Unlike Isolated I/O, this technique employs a common control line for both I/O and memory. The CPU relies on the specific addresses to determine whether the operation involves regular memory or an I/O device. This simplifies the system's design and programming, treating I/O devices as integral parts of the memory space.

Understanding these I/O mapping techniques is fundamental to designing efficient and cohesive computer architectures. The choice between Isolated I/O and Memory Mapped I/O depends on factors such as system complexity, performance requirements, and the desired level of integration between memory and I/O operations.

Data Transfer (Data Transmission)

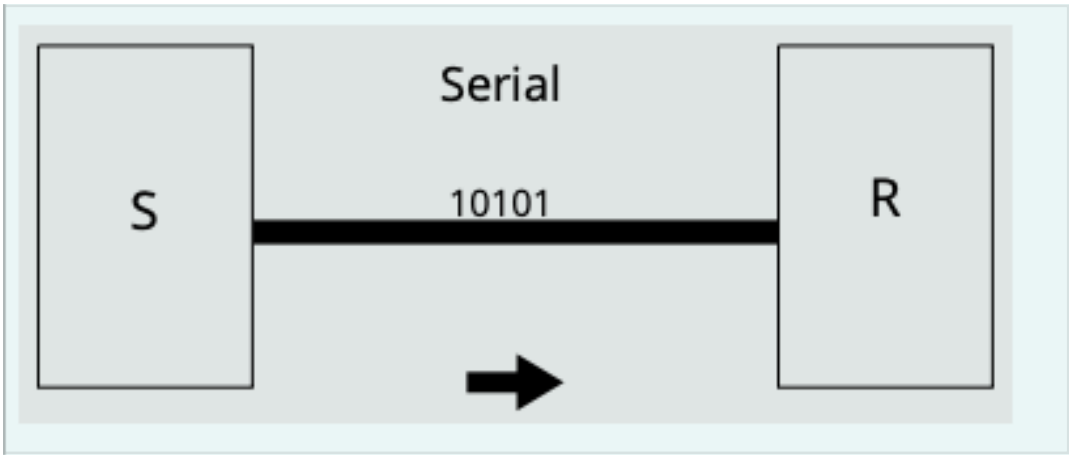
Data transfer, also known as data transmission, refers to the process of sending data from one unit to another. This fundamental aspect of computing is categorized into two main types:

1. Parallel Transmission



- Each bit in the data has its own dedicated path, allowing the entire message to be transmitted simultaneously.
- n bits must be transmitted through n separate wires, providing a faster data transfer rate.
- Although faster, parallel transmission requires a significant number of wires, making it suitable for short-distance applications where speed is critical, such as between the CPU and memory (bus) or from the CPU to a printer.

2. Serial Transmission



- Each bit of the message is sent in sequence, one at a time, through a single wire.
- n bits are transmitted using only one wire, resulting in a slower but more cost-effective data transfer process.
- Serial transmission is commonly employed for longer distances, such as communication between the CPU and I/O devices or between different computers.

Understanding the distinctions between parallel and serial transmission is crucial in designing efficient communication systems, and the choice between them depends on factors such as distance, cost considerations, and the speed requirements of the specific application.

Synchronous Data Transfer

- Two units involved in data transfer share a common clock.
- Data transfer between the sender and receiver is synchronized with the same clock pulse.
- Synchronous data transfer is typically used between devices that operate at matching speeds.
- Bits are continuously transmitted to maintain frequency synchronization between both units.
- Synchronous, in this context, means occurring at the same time, thanks to the presence of a common clock.
- This method is fast, ensuring efficient and timely data transmission.
- However, it can be costly to implement due to the requirement for a synchronized clock system.

Asynchronous Data Transfer

- Two units involved in data transfer operate independently and each has its private clock.

- Data transfer between the sender and receiver is not synchronized with the same clock pulse.
- Asynchronous data transfer is commonly used between devices that do not operate at the same speed.
- Bits are sent only when available, and the communication lines remain idle when there is no information to be transmitted.
- Asynchronous, in this context, means occurring at regular intervals due to the absence of a common clock.
- While asynchronous data transfer is slower compared to synchronous, it is more economical to implement.
- Cost-effectiveness makes it a suitable choice for scenarios where synchronization is not critical.

Understanding the characteristics of synchronous and asynchronous data transfer is crucial for designing communication systems that align with the specific requirements of different devices and applications. The choice between these methods depends on factors such as device speed compatibility, cost considerations, and the need for precise timing.

Asynchronous Data Transfer & Its Types

- Asynchronous data transfer is employed when the speed of I/O devices does not align with the processor, and the timing characteristics of the I/O device are unpredictable.
- **Definition:** Asynchronous data transfer between two independent communicating units requires the transmission of control signals. These signals indicate the time at which data is being transmitted and ensure proper coordination between the communicating units.
- Asynchronous data transfer is implemented using two distinct methods:
    1. **Strobe Control:** In this method, the sender informs the receiver of an impending data transfer by sending a strobe pulse. However, a challenge with this approach is that it merely indicates that data is on the way, without confirming whether the receiver has successfully received it. To address this limitation, a newer method called the handshaking method was introduced.
    2. **Handshaking Method:** In this method, when the sender intends to send data to the receiver, it first notifies the receiver of the upcoming data transfer. In response, the

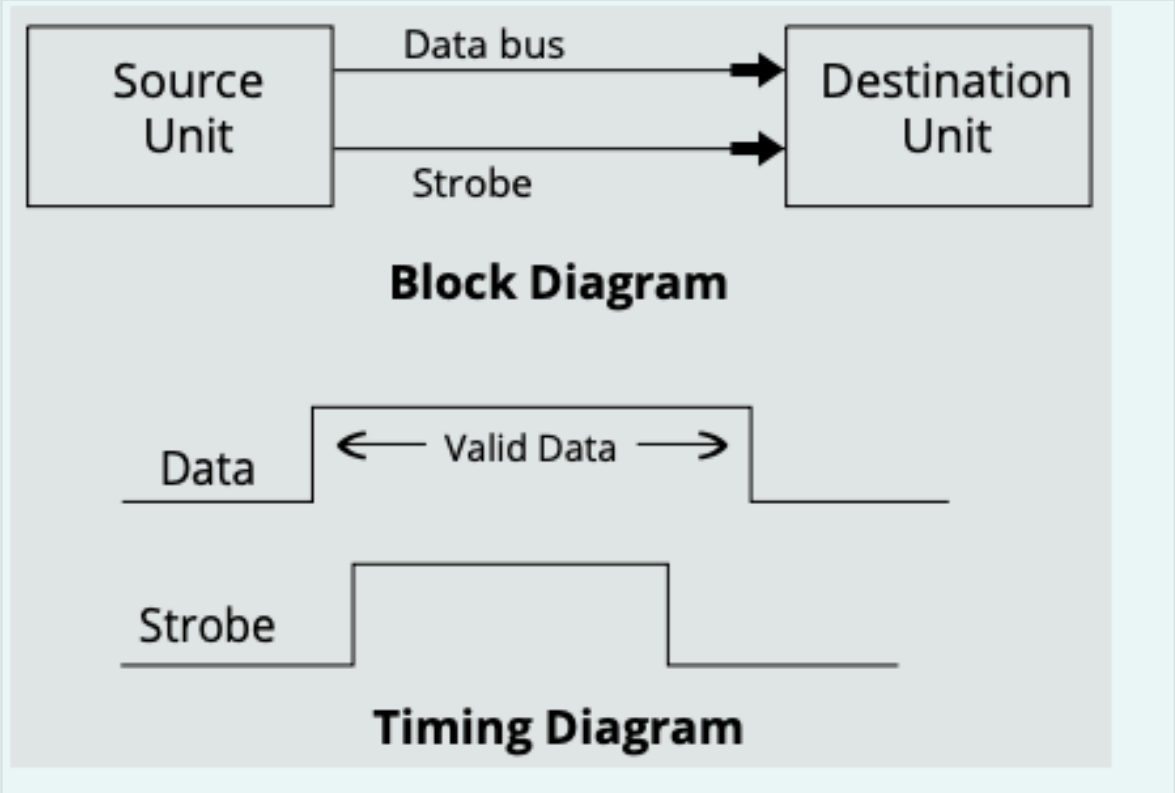
receiver sends an acknowledgment signal, confirming its readiness to receive the data. This two-way confirmation ensures a more reliable and secure data transfer process compared to the strobe control method.

Asynchronous data transfer, with its reliance on control signals and handshaking methods, offers a flexible and robust solution for scenarios where the speed and timing characteristics of communicating devices may vary. The choice between strobe control and handshaking methods depends on factors such as the reliability and confirmation requirements of the data transfer process.

Strobe Control

- Strobe control employs a single control line to time each data transfer.
- The strobe signal may be activated by either the source or the destination unit.

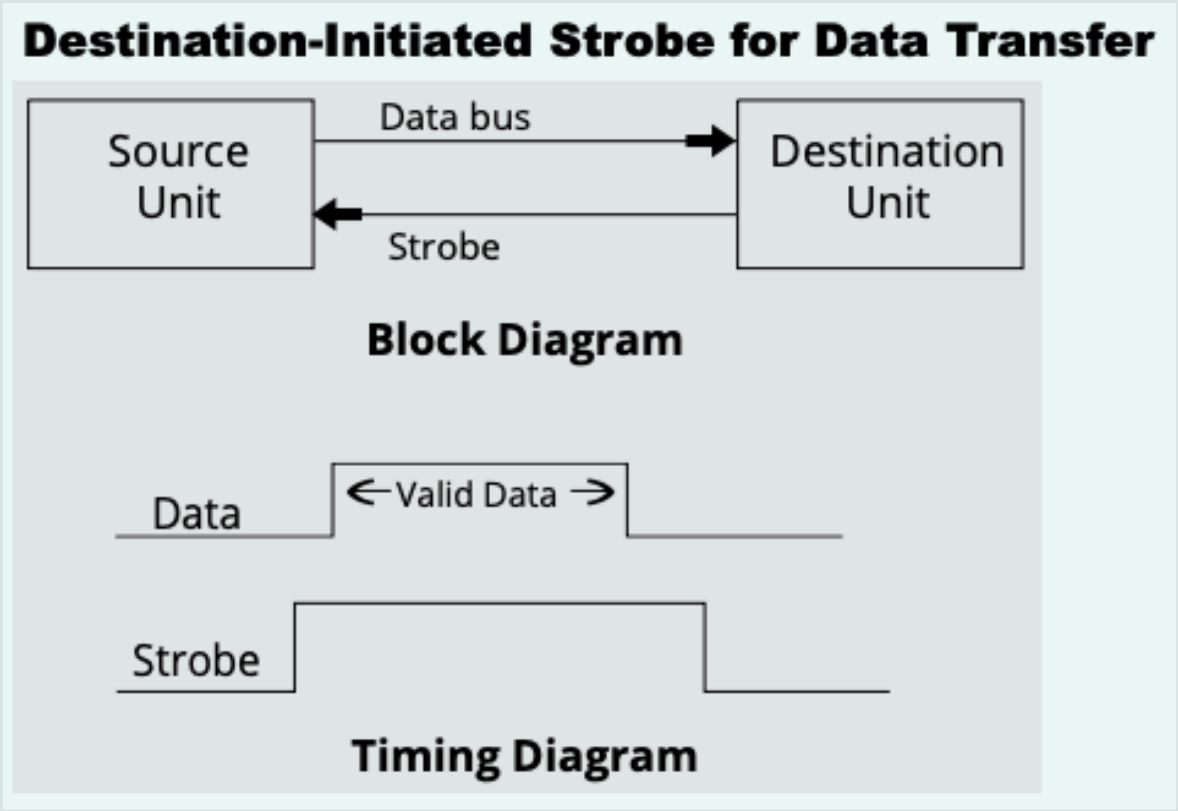
Source-Initiated Strobe for Data Transfer



- In this scenario, there are a source unit and a destination unit, along with two lines: a data bus line for data transfer and a strobe line for signaling.
- The timing diagram illustrates that the data is loaded first, followed by the transmission of the strobe signal, indicating the completion of data transfer.

- **Example:** A memory write control signal from the CPU to the memory unit. In this case, the CPU initiates the transfer by loading data onto the bus and sending the strobe signal to inform the memory unit.

Destination-Initiated Strobe for Data Transfer



- In this configuration, there is a source unit and a destination unit. However, the destination unit requests data, initiating the process by sending the strobe signal before the actual data transfer.
- **Example:** A memory read control signal from the CPU to the memory unit. Here, the memory unit requests data by sending the strobe signal to the CPU, prompting the data transfer.

Strobe control, whether source-initiated or destination-initiated, provides a method for precisely timing data transfers between units. Understanding these mechanisms is crucial in designing reliable communication systems where data synchronization and integrity are paramount.

Disadvantages of Strobe Control

- One significant drawback of strobe control is that the source unit initiating the transfer lacks confirmation of whether the destination unit has successfully received the data item placed on the bus.

- Conversely, when the destination unit initiates the transfer, it has no means of knowing whether the source unit has indeed placed the data on the bus as intended.
- This lack of acknowledgment introduces a potential challenge in ensuring the reliability and success of data transfer operations.

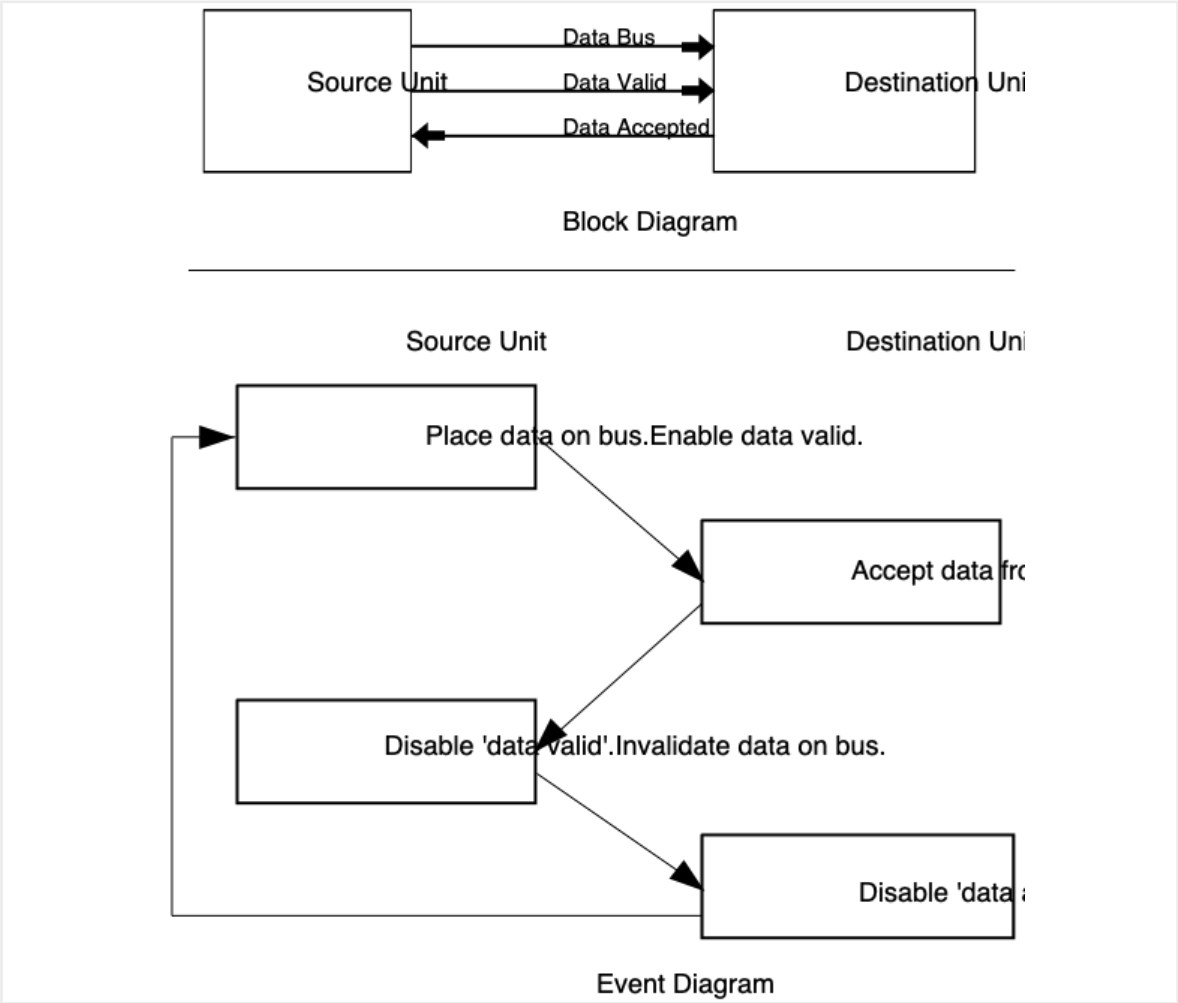
While strobe control provides a simple and efficient means of timing data transfers, the absence of acknowledgment poses a limitation in guaranteeing the integrity of the communication process. Overcoming this limitation often requires the adoption of more sophisticated techniques, such as handshaking methods, to ensure a two-way confirmation of successful data transmission between communicating units.

- Note: In general, communication between the CPU and I/O Interface is typically achieved using strobe control. On the other hand, when there is communication between an I/O device and the I/O interface, it is commonly done using handshaking methods. This strategic choice in communication techniques ensures an efficient and reliable data transfer process tailored to the specific requirements and interactions between the central processing unit and the input/output components.

Handshaking Method

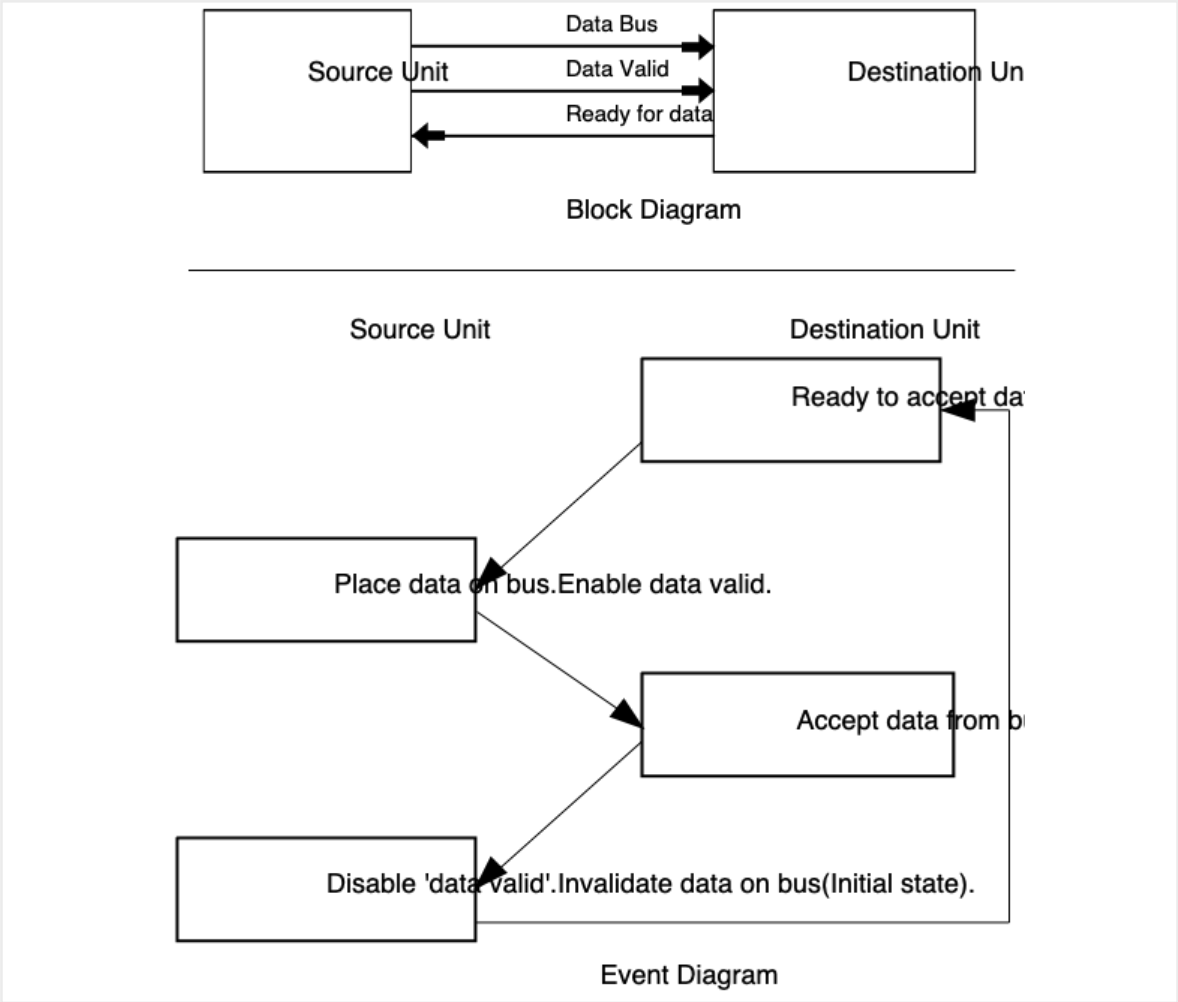
- The handshaking method addresses the limitations of the strobe method by introducing a second control signal that provides a reply to the unit initiating the transfer.
- Combining strobe control with an acknowledgment signal results in a two-wire control system, enhancing the reliability of data transfer.
- In this method, three lines connect the source unit and the destination unit: the data bus, the data valid line (indicating data initiation or transfer), and the data accepted line (providing a reply to the initiation).

Source-Initiated Handshaking



- This configuration involves two control signals: the data valid signal, indicating when the source unit initiates data transfer, and the data accepted signal, the reply from the destination unit.
- Sequence of operation: The data is placed on the data bus first, followed by the source unit initiating the data valid signal. After successfully receiving the data, the destination unit responds with a signal through the data accepted line to the source unit. Subsequently, the data valid signal is disabled, and the destination unit disables the data accepted signal.

Destination-Initiated Handshaking



- In this scenario, the destination unit initiates data transfer by sending the "ready for data" signal to the source unit, indicating its readiness to accept data.
- Following this signal, the source unit places the data on the data bus and enables the data valid signal. After enabling, the destination unit accepts the data and disables the "ready for data" signal. Finally, the source unit disables the data valid signal, returning the system to its initial state.

The handshaking method, with its two-wire control system and bidirectional communication, ensures a more robust and synchronized data transfer process. Whether initiated by the source or the destination, this method provides reliable confirmation and control signals, enhancing the overall integrity of the communication between units.

Modes of Transfer

- Before delving into the modes of transfer, it's essential to understand that when data is transferred between the CPU and I/O devices, the CPU primarily handles execution and stores some data in its registers. However, the actual data

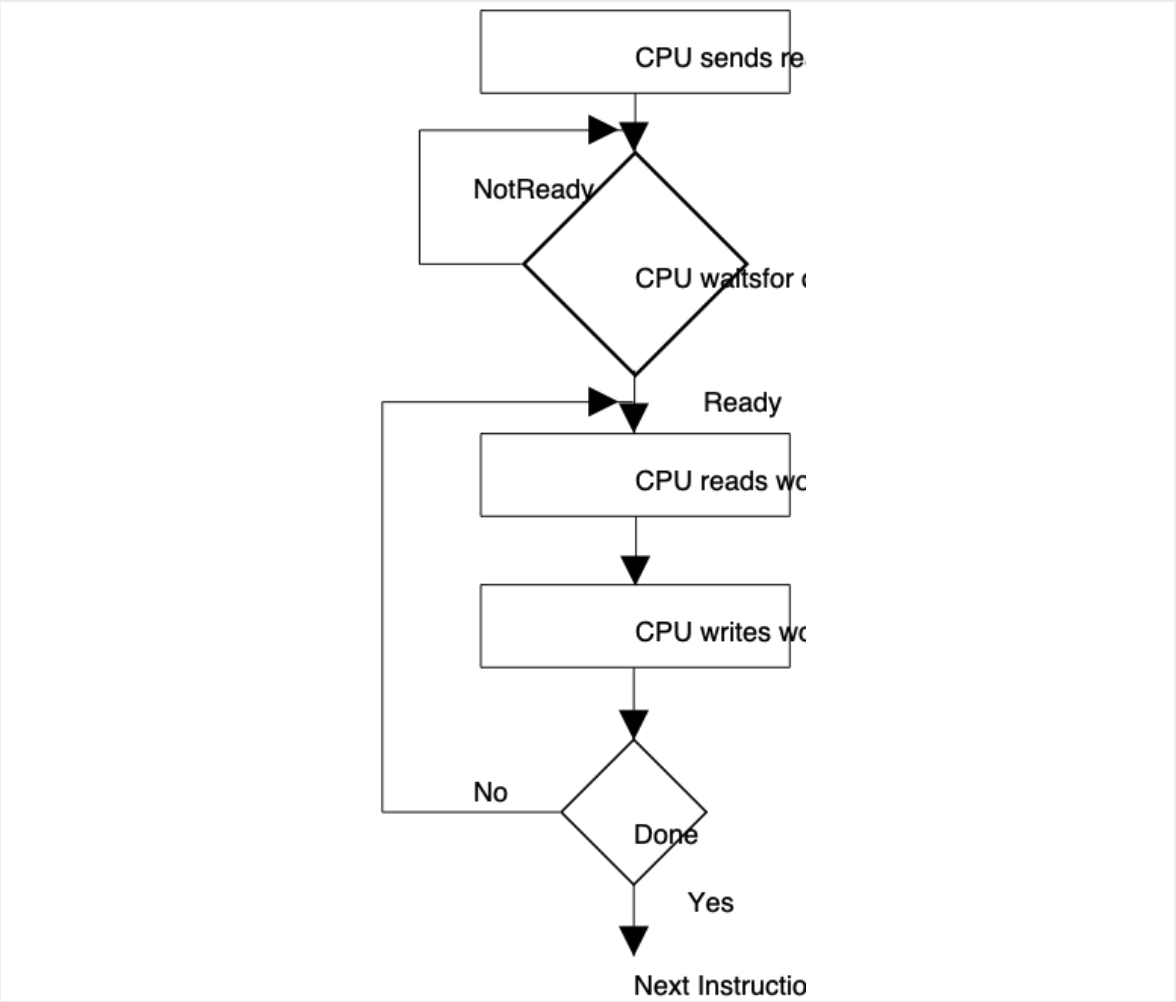
transfer to I/O devices is facilitated through the assistance of memory. Therefore, memory plays a crucial role, serving as an intermediate step when performing data transfers from I/O devices. In this process, the CPU acts as an intermediary.

- Various modes exist for transferring data between the central computer and I/O devices, and understanding these modes is integral to comprehending the intricacies of data transfer.
- There are three primary modes through which data is transferred from peripheral devices to the CPU:
    1. **Programmed I/O:** In this mode, the CPU serves as an intermediate link in the data transfer process.
    2. **Interrupt-Initiated I/O:** Similar to Programmed I/O, this mode involves the CPU as an intermediary in the data transfer.
    3. **Direct Memory Access (DMA):** In DMA mode, the CPU is not an intermediate participant, allowing for more efficient data transfer between peripheral devices and memory.
- While Programmed I/O and Interrupt-Initiated I/O involve the CPU as an intermediate step, Direct Memory Access (DMA) bypasses the CPU, streamlining the data transfer process.

Programmed I/O

- Programmed I/O is utilized when there is a need to transfer data between the CPU and I/O, and the data transfer is managed through program instructions.
- In the context of computer programming languages such as C++, Programmed I/O operations result from I/O instructions embedded in the computer program. Examples of such instructions include those for input (e.g., cin) and output (e.g., cout).
- Each data transfer in Programmed I/O is initiated by an I/O instruction within the program, typically to access registers or memory on a specific device.
- Executing data transfers under program control demands continuous monitoring of I/O devices by the CPU.

The diagram below illustrates the process of Programmed I/O:



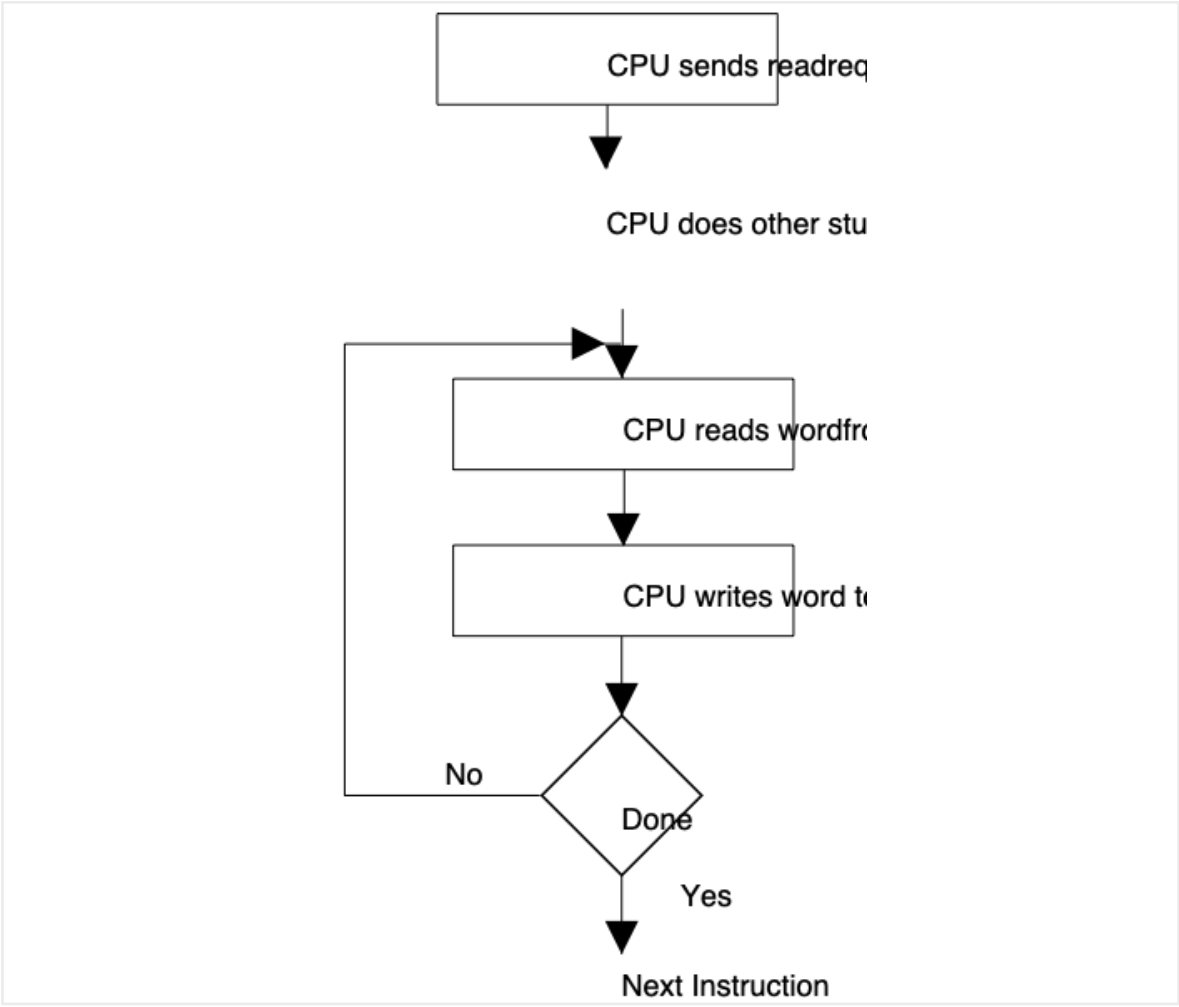
In Programmed I/O, the CPU initiates a request and then remains in a program loop (polling) until the I/O device signals its readiness for data transfer. Importantly, the I/O device takes no further action to interrupt the CPU; it does not independently interrupt the CPU.

- Disadvantages:
  - Programmed I/O can be time-consuming as it keeps the CPU unnecessarily busy. To address this issue, interrupt facilities are often employed to enhance efficiency.

Interrupt Initiated I/O

- Interrupt Initiated I/O was introduced to address the polling-related issue present in Programmed I/O.
- An interrupt is a high-priority signal, generated either by an external device or some software, designed to immediately capture the CPU's attention. The use of interrupts aims to eliminate the waiting period inherent in Programmed I/O.
- In Interrupt Initiated I/O, instead of the CPU continuously monitoring, the interface is informed to issue an interrupt request signal when data becomes available from the device.
- Meanwhile, the CPU proceeds to execute other programs

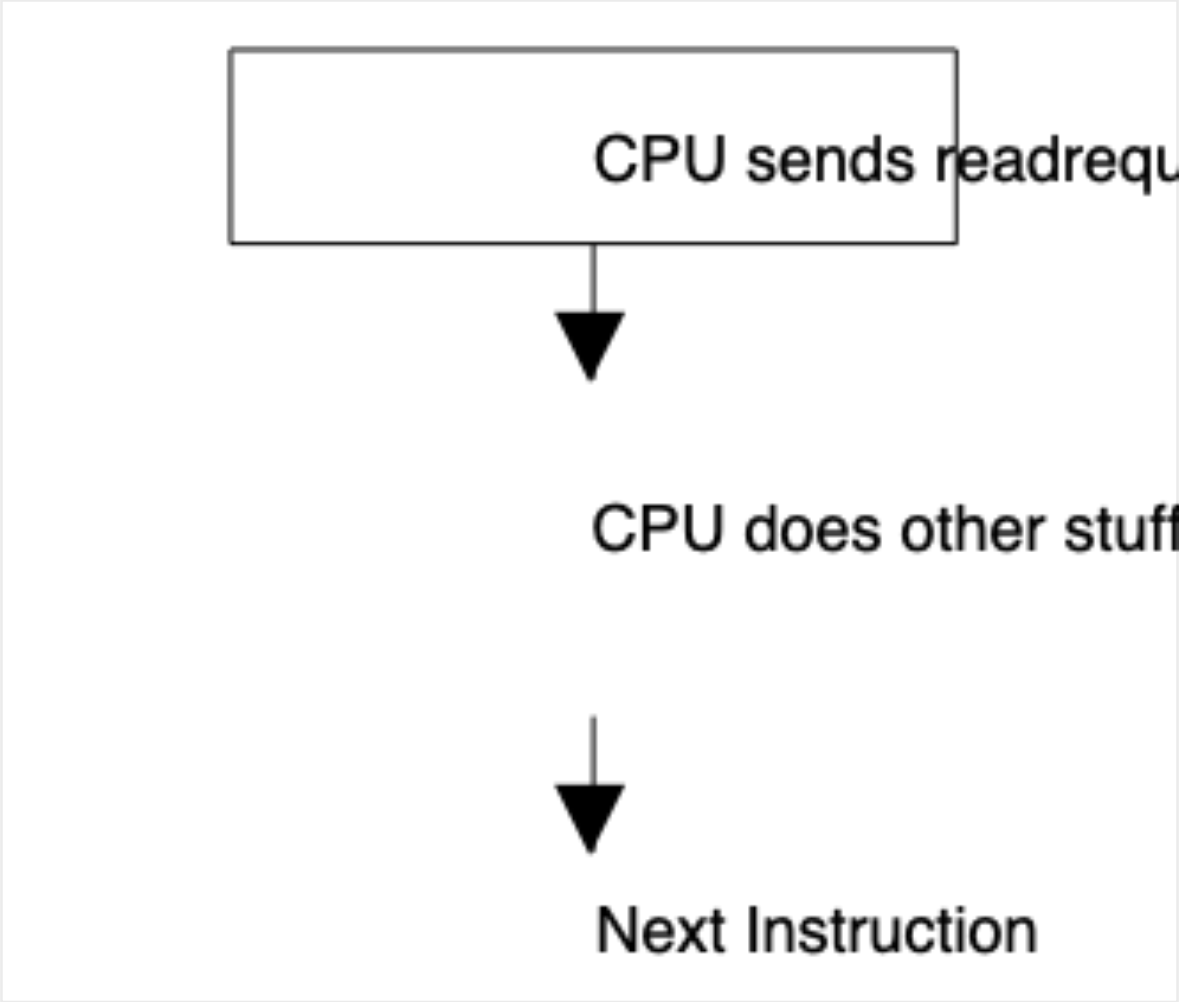
while the interface keeps monitoring the device.
- When the device is ready for data transfer, it generates an interrupt request.
- Upon detecting the external interrupt signal, the CPU interrupts its current task, processes the I/O data transfer, and then resumes the original task it was performing.



Direct Memory Access (DMA)

- DMA is employed when large blocks of data need to be transferred between the CPU and I/O devices, rendering Programmed or Interrupt Initiated I/O less efficient.
- For high-speed transfers of substantial data blocks between external devices and main memory, the DMA approach is often utilized.
- In other transfer modes, memory is accessed indirectly through the CPU. However, when transferring significant data blocks and frequently utilizing the CPU before memory access, this process becomes time-consuming. DMA addresses this by allowing direct communication between I/O devices and memory, minimizing CPU intervention.

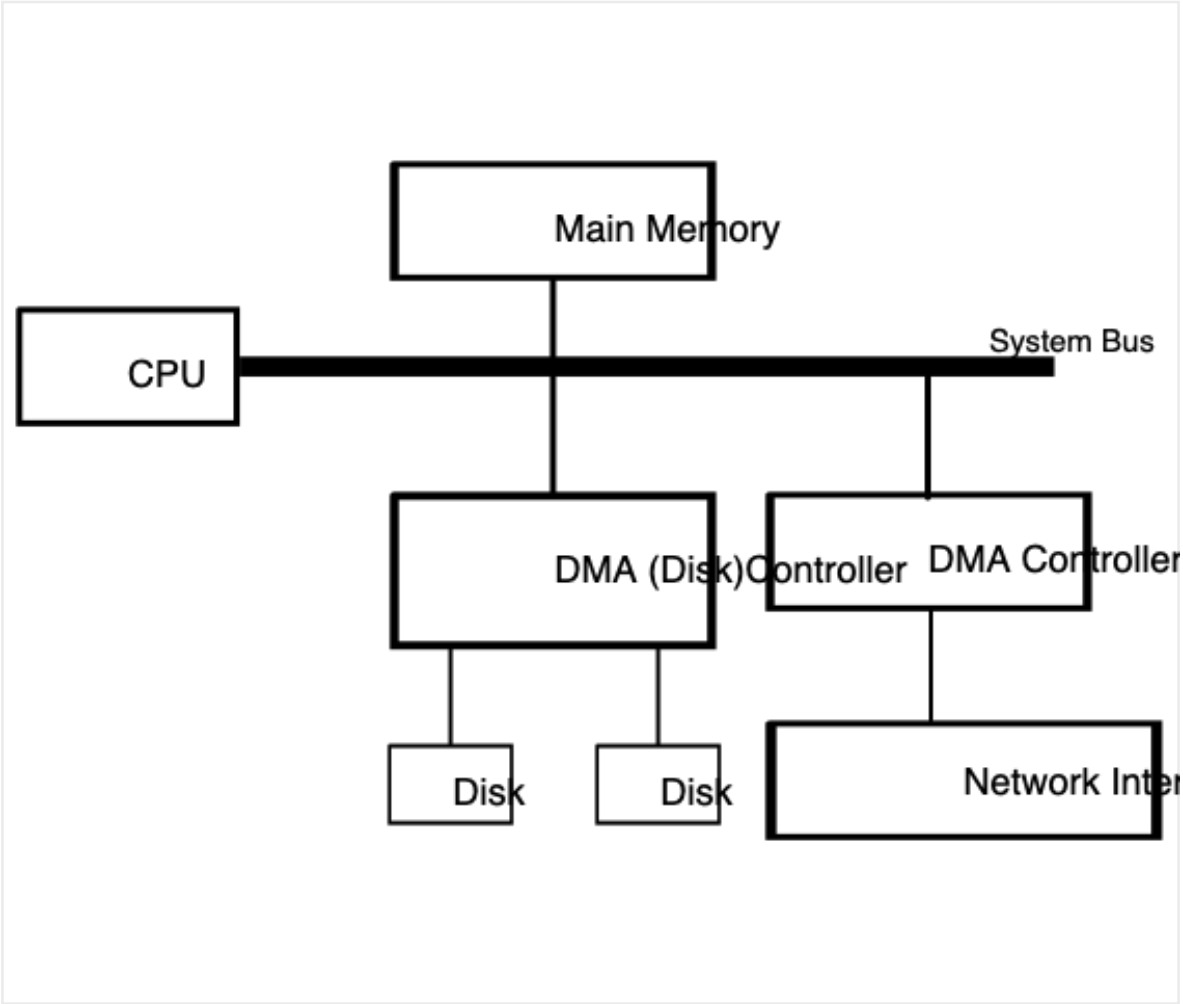
- DMA permits data transfer directly between the I/O device and main memory with minimal CPU involvement.
- In DMA, the CPU grants the I/O interface the authority to read from or write to memory without direct CPU intervention.
- The DMA controller autonomously manages data transfer between main memory and the I/O device.
- The CPU is only involved at the beginning and end of the transfer and is interrupted only after the entire block has been successfully transferred.



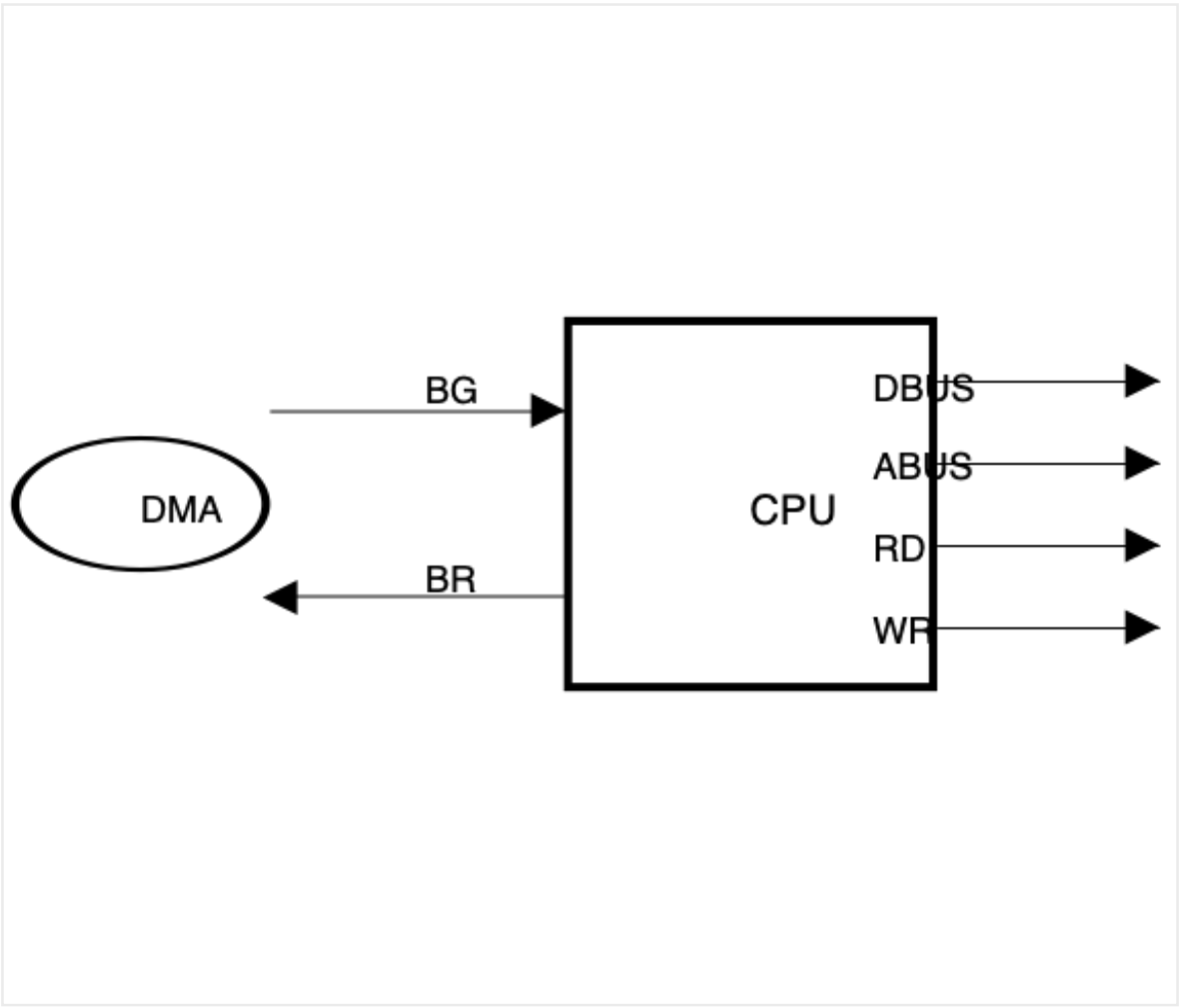
- The process involves the CPU initiating the DMA controller to transfer data between a device and main memory, allowing the CPU to proceed with other tasks.
- The DMA controller issues a request to the relevant I/O device, manages the data transfer between the device and main memory, and waits for its completion.
- Upon the conclusion of the data transfer, the DMA controller interrupts the CPU.

DMA Controller

- DMA enables I/O devices to transfer data directly to or from main memory without requiring CPU intervention, effectively bypassing the CPU.
- Between I/O devices and the CPU, there exists an interface, but between memory and I/O devices, the DMA controller (or DMA channel) acts as the intermediary, creating a channel between main memory and I/O devices. Devices such as magnetic disks, USB drives, network cards, graphics cards, and sound cards, when connected via DMA, can achieve faster data transfer rates.
- DMA finds applications in systems utilizing multicore architectures, particularly in scenarios where intrachip data transfers are required.
- Within the system bus, comprising address lines, data lines, and control lines connected to the CPU, memory, and DMA controller, during DMA transfers, the CPU is temporarily disabled. While the CPU typically controls the system bus, in DMA transfers, the DMA controller temporarily borrows control of the system bus from the CPU to facilitate efficient data transfer between I/O devices and memory.



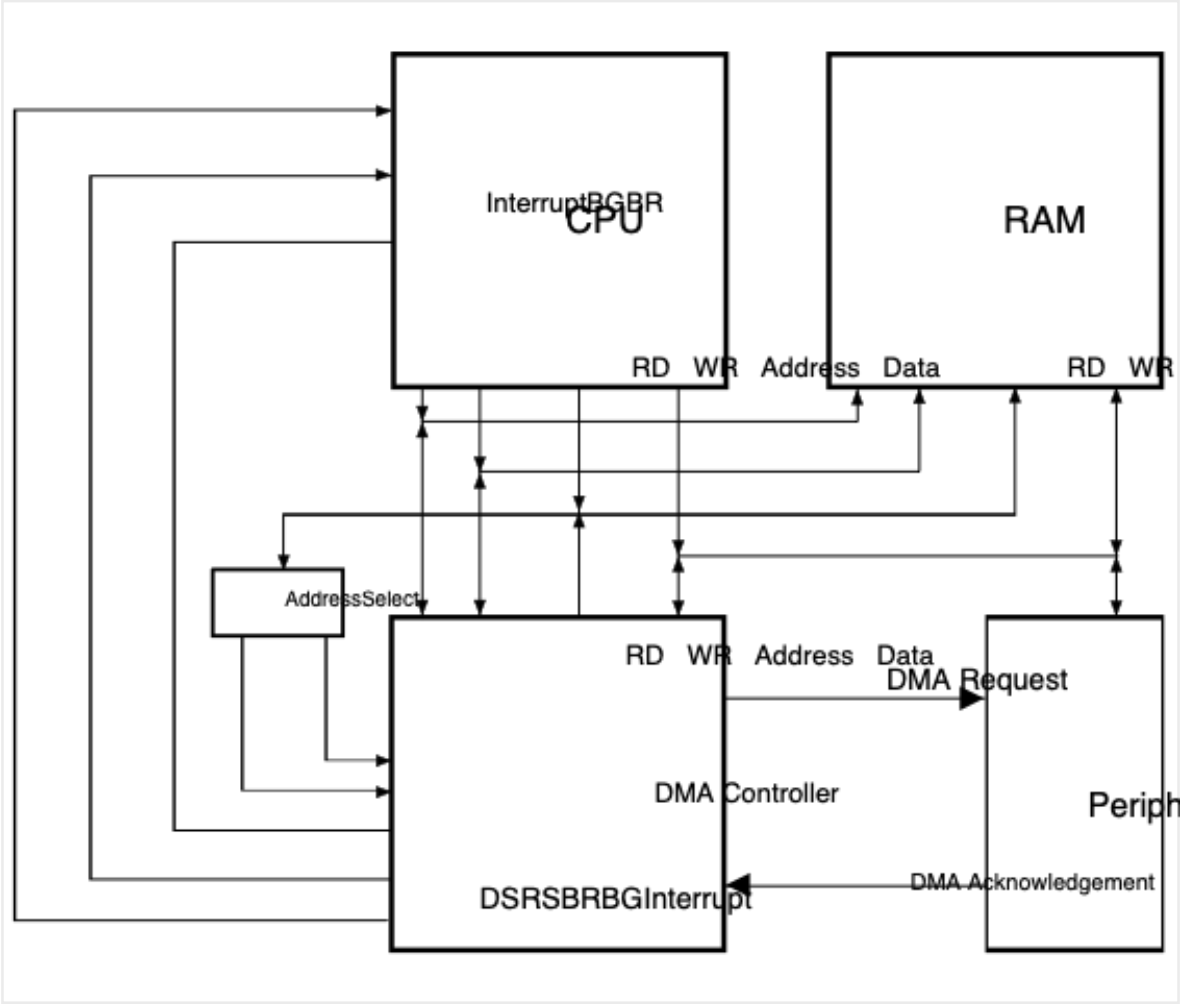
How DMA makes CPU to go in idle state?



- To facilitate DMA's control over the bus system from the CPU, two signals are employed: Bus Request (BR) and Bus Grant (BG).
- When DMA desires full control of the bus system, it initiates the process by sending a Bus Request (BR) signal through the bus request line.
- Upon receiving the Bus Request (BR) signal, the CPU interrupts its ongoing tasks, relinquishes control of all three components—data lines, address lines, and control lines—and enters a high-impedance state. In this state, the bus behaves like an open circuit, disabling all signals and buses.
- To signal to the DMA that control has been transferred, the CPU sends a Bus Grant (BG) signal through the bus grant line, indicating that the DMA now has authority over the buses. This communication allows the DMA to use the buses to transfer data directly to memory.
- Essentially, the two main signals used in this process are Bus Request (BR) and Bus Grant (BG).

DMA Working

When an I/O device wishes to transfer data with main memory, the following steps outline the DMA process:



In the diagram above, we have a detailed breakdown of the components involved in the Direct Memory Access (DMA) process:

1. **DS (DMA Select):** The processor sets DS = 1 to activate DMA, initiating the DMA process.
2. **RS (Register Select):** The CPU uses this signal to select DMA registers for storing values, such as the starting address and the number of words to be transferred.
3. **RD (Read) & WR (Write):** These signals, Iri is used for reading and writing purposes during the DMA operation.
4. **BR (Bus Request):** DMA employs this line to send a request to the processor to release the BUS system, indicating its need for control over the system bus.
5. **BG (Bus Grant):** When the processor relinquishes control of the bus to DMA, it sets BG = 1, signifying that the bus is now granted to the DMA controller.
6. **Interrupt:** DMA uses this line to send a signal when data transfer is completed. The processor can also use this line to

check whether data transfer has been successfully completed.

7. **DMA Request:** I/O devices utilize this line to send a request to the DMA controller, signaling the need for data transfer.
8. **DMA Acknowledgement:** The DMA controller responds to I/O devices through this line, acknowledging the receipt of the request and preparing for data transfer.
9. **Address Register:** The processor stores the starting address of data in this register, providing the necessary information for the DMA controller to locate the data.
10. **Word Count Register:** The processor stores the total number of words to be transferred in this register, allowing the DMA controller to determine the extent of the data transfer operation.
11. **Control Register:** The processor stores control signals in this register, dictating various aspects of the DMA operation, such as the transfer mode and direction.
12. **Data Bus Buffer:** This component is employed to temporarily store data during the DMA transfer process, ensuring efficient and synchronized data movement.
13. **Data Bus:** DMA utilizes this bus for the actual transfer of data between memory and peripheral devices, facilitating high-speed and direct communication.

Now, let's delve into the operational sequence of the Direct Memory Access (DMA) process:

- **Initiation of Data Transfer Request:** When an I/O device wishes to transfer data, it sends a request to the DMA through the DMA request line.
- **Bus Request to Processor:** DMA, in response to the request, sends a bus request (BR = 1) to the processor, requesting it to release control of the bus system (utilizing the BR line).
- **Processor's Response:** Upon receiving the bus request, the processor stores its current work, and then transmits essential information, such as the starting address and the number of words to be transferred, to DMA. Subsequently, the processor relinquishes control of the bus system and notifies DMA by setting the BG line to 1.
- **DMA Acknowledgement and Data Transfer:** Upon receiving the BG signal, DMA activates the DMA acknowledgement line, informing the I/O device that it can commence data transfer.

The DMA controller initiates the actual data transfer process.

- **Decrementing Word Count:** With each data transfer, the value of the Word Count (WC) register is decremented by 1, keeping track of the progress of the data transfer operation.
- **Data Transfer Completion:** When the WC register reaches 0, DMA sets BR = 0 and sends an interrupt signal to the CPU, signaling the completion of the data transfer.
- **CPU's Post-Transfer Actions:** The CPU, upon receiving the interrupt, checks the WC register. Since it is now 0, the CPU sets BG = 0, reclaiming control of the bus system for its own operations.

Input Output Processor (IOP)

- The IOP is an external processor designed to handle input and output tasks, communicating directly with all I/O devices.
- It operates independently of the CPU and relieves the CPU of input-output responsibilities.
- Similar to a CPU, the IOP can fetch and execute its own instructions, perform arithmetic, logic, branching, and code translation.
- The IOP provides a data transfer path between peripheral devices and the memory unit.
- Unlike DMA controllers, the IOP can fetch and execute instructions, making it versatile in handling various processing tasks.

Data Formatting and Transfer

- Peripheral devices often have different data formats than memory and CPU. The IOP is responsible for structuring data words to match the required formats.
- For example, it may receive 4 bytes from an input device and pack them into one 32-bit word before transferring to memory.

CPU-IOP Communication:

- In most systems, the CPU is the master, and the IOP is a slave processor.
- The CPU initiates all operations, but I/O instructions are executed by the IOP.
- Communication involves a sequence of operations:
  1. The CPU sends an instruction to test the IOP path.
  2. The IOP responds by sending its status.
  3. The CPU sends the instruction to start I/O transfer by

specifying the memory address where the IOP should begin.
4. The CPU can proceed the other tasks while the IOP handles the I/O program.
5. After data transfer completion, the IOP sends an interrupt request to the CPU.
6. The CPU reads the IOP status, with the IOP placing the status report into a designated memory location.

Serial Communication

- A data communication processor is an I/O processor designed for communication with remote terminals through data communication networks.
- It handles tasks like distributing and collecting data from various devices connected through communication lines.
- The computer appears to serve many users simultaneously in a time-sharing environment by interspersing fragments of each network demand efficiently.
- Unlike I/O processors communicating through a common bus, data communication processors communicate with each terminal through a single pair of wires.
- Data and control information are transferred serially, resulting in a slower transfer rate compared to common bus communication.
- The data communication processor communicates with CPU and memory similar to any I/O processor.

Connecting Remote Terminals:

- Remote terminals connect to a data communication processor via telephone lines or other communication facilities.
- Conversion devices like data sets, acoustic couplers, or modems are used to convert digital signals to audio tones for transmission over telephone lines.
- Different modulation schemes, communication media, and transmission speeds are employed.

Transmission Methods:

- Communication lines may be connected to synchronous or asynchronous interfaces based on the remote terminal's transmission method.
- Asynchronous transmission uses start and stop bits in each character, while synchronous transmission sends a

continuous message without start-stop bits.
- Synchronous transmission is more efficient but requires continuous messages to maintain synchronism.

Error Detection

- Data communication processors check for transmission errors using methods like parity checking in asynchronous transmission and techniques like longitudinal redundancy check (LRC) or cyclic redundancy check (CRC) in synchronous transmission.
- LRC checks are calculated at the end of a block, and the receiving station compares it with the transmitted LRC.

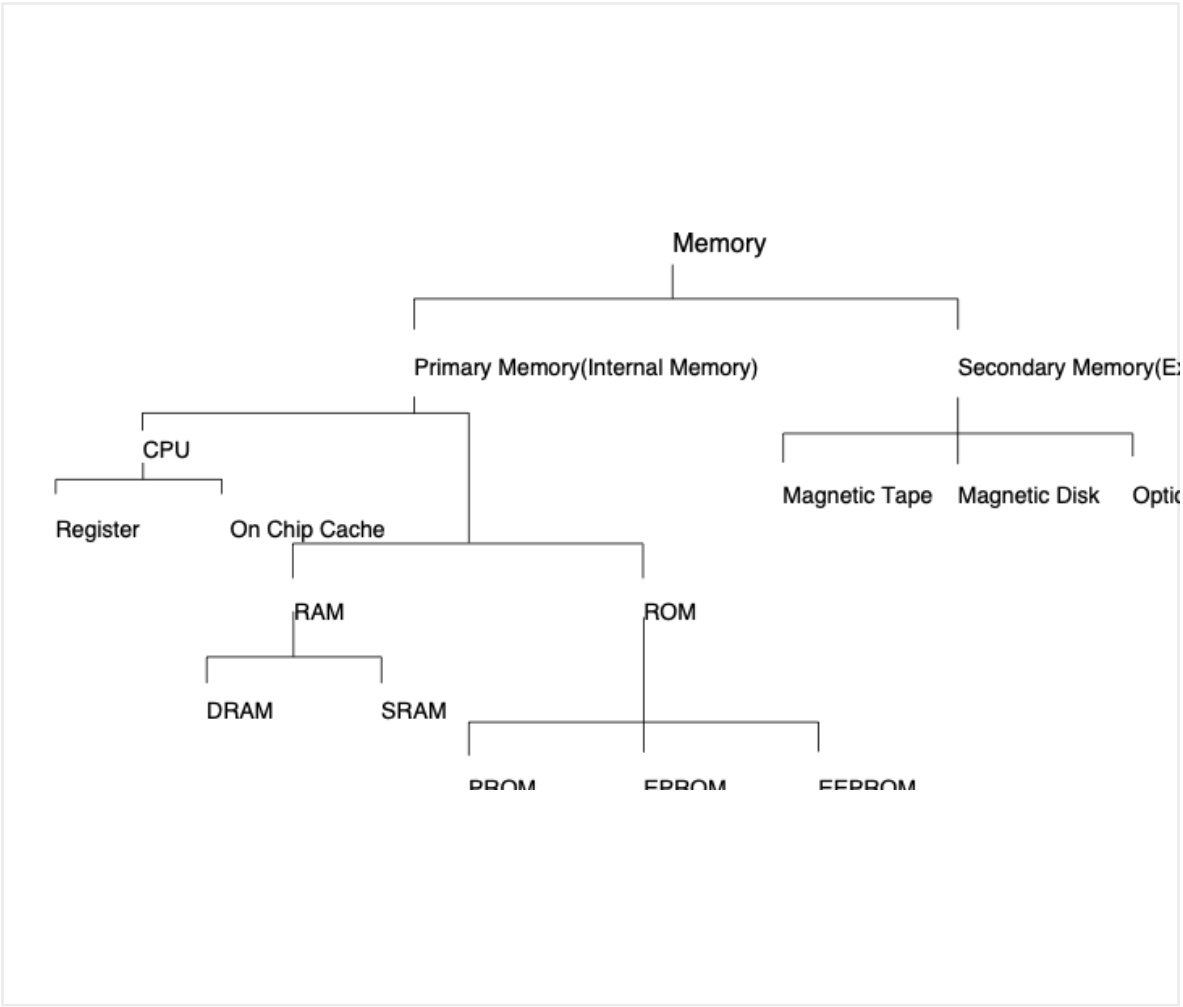
Transmission Modes:

- Data can be transmitted in three modes: simplex (one-way communication), half-duplex (two-way, but one direction at a time), and full-duplex (simultaneous two-way communication).
- Simplex is rarely used in data communication. Half-duplex requires a turnaround time, and full-duplex can use either a four-wire link or frequency spectrum subdivision in a two-wire circuit.

Data Link and Protocols

- The communication lines, modems, and equipment form a data link, and orderly data transfer is governed by a protocol.
- Data link control protocols ensure the orderly transfer of information, establish and terminate connections, identify sender and receiver, handle error-free message passing, and manage control functions.
- Protocols are categorized into character-oriented and bit-oriented protocols based on the framing technique used.

This unit delves into the fundamental aspects of computer memory within the context of Computer Architecture and Organization. Memory, a vital component, significantly influences the efficiency and functioning of computers. The topics covered include the Classification of Memories, exploring RAM organization, Static RAM, and Dynamic RAM. Additionally, we will examine ROM organization, including PROM, EPROM, EEPROM, and EAPROM. The concept of Memory Hierarchy will be discussed, shedding light on Cache Memory, Mapping techniques, and the importance of the locality of references. Advanced topics such as Virtual Memory will also be explored, focusing on demand paging, Page Faults, and Page Replacement strategies.

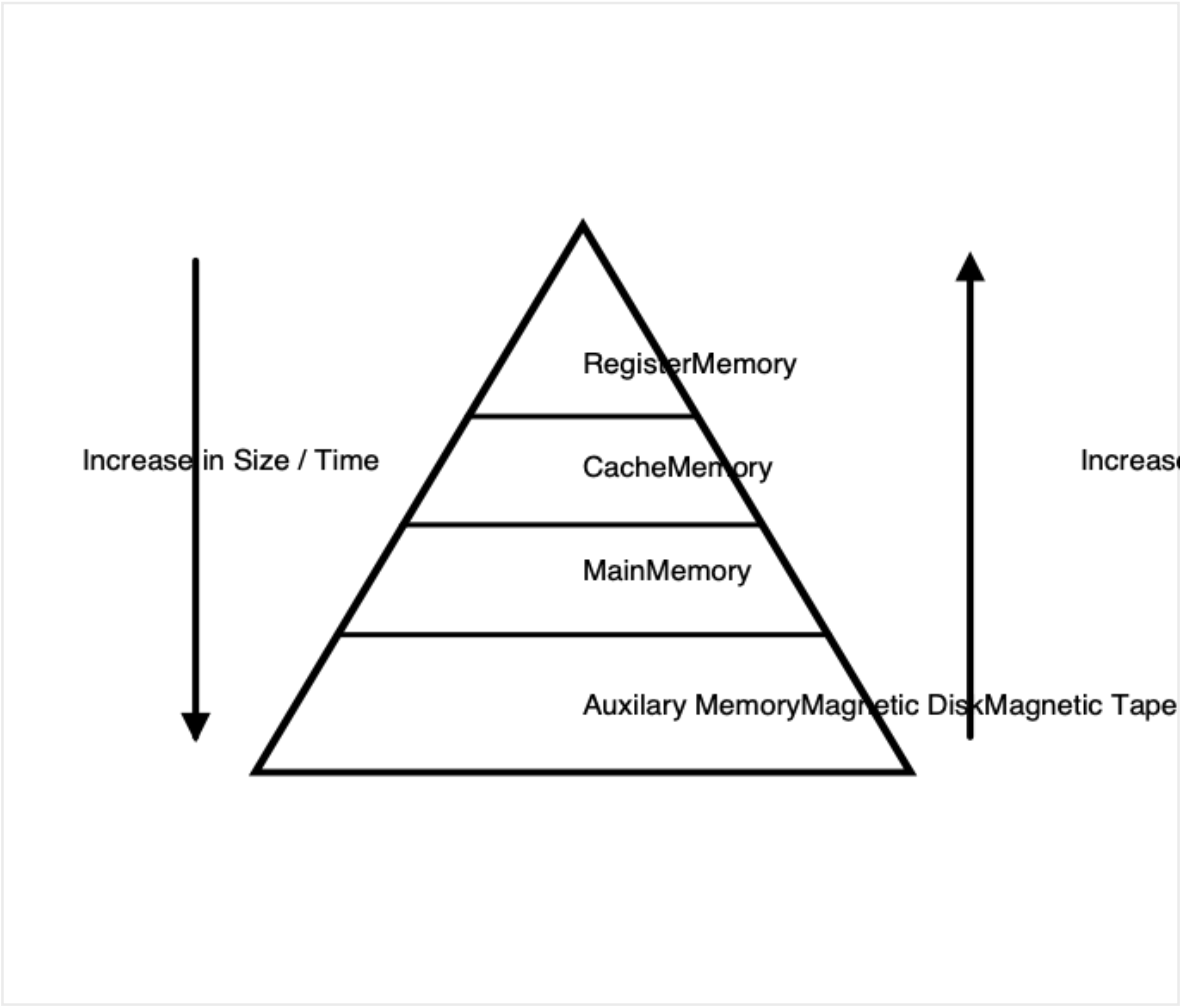


- Random Access Memory (RAM), Read-Only Memory (ROM), Cache memory, and processor registers collectively constitute the primary memory of a computer system. Primary memory, also known as main memory, plays a pivotal role as it is directly accessed by the Central Processing Unit (CPU). This central storage area is characterized by its speed and serves as the primary repository for programs and data within a computer architecture.
- RAM, being volatile memory, allows for quick read and write operations, facilitating the temporary storage of actively used programs and data. ROM, on the other hand, is non-volatile and retains information even when the power is turned off. It typically contains firmware and essential system instructions that are crucial for the computer's basic functions.
- Cache memory acts as an intermediary between the main memory and the CPU, storing frequently accessed data and instructions to expedite processing speed. Processor registers are small, high-speed storage locations within the CPU itself, used to store and manage crucial data for

immediate processing.

- The seamless interaction between these components ensures efficient data retrieval and execution, contributing to the overall performance and responsiveness of the computer system. As a cohesive unit, RAM, ROM, Cache memory, and processor registers collectively form the backbone of the main memory hierarchy, playing a critical role in the seamless functioning of modern computing devices.

Memory Hierarchy



Memory is a fundamental component of digital computers, serving as the repository for data and programs. However, relying on a single type of memory poses limitations on the storage capacity and access speed. To address this, computer systems employ a concept known as Memory Hierarchy, which encompasses various memory levels to optimize performance.

- Using only one type of memory, such as RAM, is impractical for storing all programs and data, and relying solely on secondary storage (e.g., hard disk) leads to slow access times by the CPU. The solution lies in employing multiple

memory levels, combining two or three types of memory to form the memory hierarchy of a digital computer.
    - Memory Hierarchy is a pyramid-like structure that visually represents the different types of memory in a computer.
    - In this hierarchy, various memories, including auxiliary, main, cache, and registers, are positioned at different levels.
    - Starting from the bottom, auxiliary memory, represented by magnetic disks and tapes, is slower but offers high capacity.
    - Main memory, located above auxiliary memory, is relatively faster and communicates directly with the CPU. It stores data and programs required for execution.
    - Cache memory, positioned above main memory, is smaller but faster. It directly supports the CPU by holding the currently executed programs and data segments.
    - Registers, located inside the CPU, are the smallest and fastest memory components directly attached to the processor.
- Comparing these memory levels, a pattern emerges. As we move from top to bottom, the size of the memory increases (registers being the smallest), but the access time also increases, indicating slower access speeds. Conversely, moving from bottom to top in the hierarchy, the speed of access increases along with the cost of the memory.
- Understanding Cache Memory:
    - The utilization of cache memory is crucial in computer systems due to the inherent speed difference between the main memory and the CPU registers. The primary purpose of cache memory is to bridge this speed gap and enhance overall system performance.
    - When the CPU needs to access data or instructions, it first checks whether the required information is available in the cache memory. Cache memory is smaller in size but significantly faster than the main memory. It stores frequently accessed data and instructions, aiming to provide the CPU with quick and efficient access to the most relevant information.
    - The cache operates on the principle of temporal and spatial locality of reference, meaning that it stores

recently accessed data and anticipates future data needs based on the program's execution patterns. By doing so, cache memory minimizes the time the CPU spends waiting for data, ultimately reducing latency and enhancing the overall speed of data retrieval.
    - Cache memory acts as a buffer between the high-speed registers of the CPU and the slower main memory. When the CPU requires data, it first checks the cache. If the required data is found in the cache (a cache hit), it can be retrieved quickly. In case of a cache miss, where the required data is not in the cache, the CPU fetches the data from the larger and slower main memory.
    - Overall, cache memory is a pivotal component in modern computer architecture, contributing to the efficient and swift operation of processors by optimizing the data access process and mitigating the performance gap between the CPU and main memory.

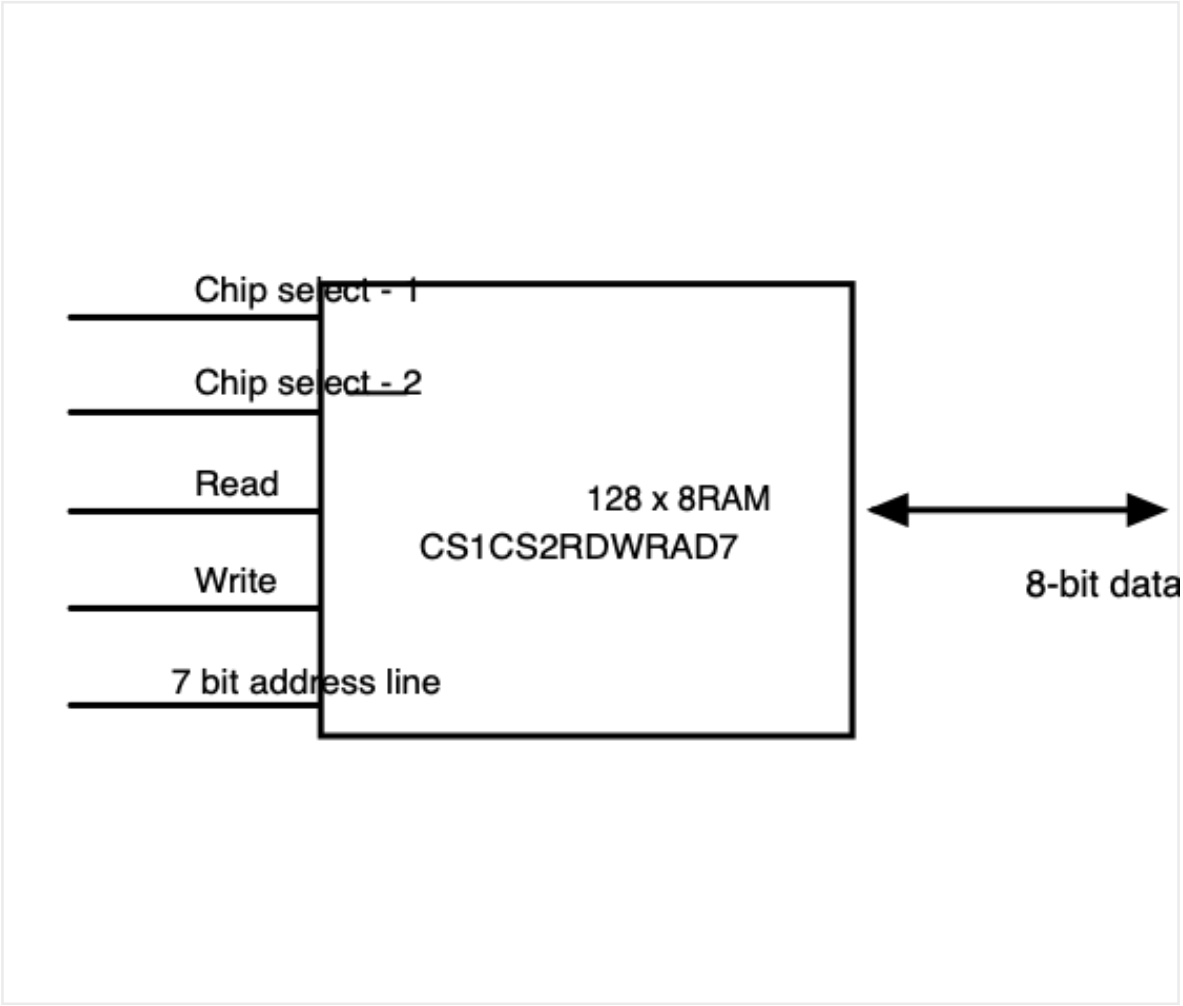
RAM Organization

- RAM, which stands for Random Access Memory, is a type of volatile memory. Volatility means that the data stored in RAM is lost when the computer is switched off. The term "random access" implies that data in RAM can be accessed in any order, making it a versatile and fast form of memory.
- During the computer's startup process, the operating system is loaded from secondary storage (such as a hard disk drive or solid-state drive) into RAM. Similarly, when an application is launched, it is loaded into RAM. This is because accessing data from RAM is over a hundred times faster for the CPU compared to accessing it from secondary storage. RAM is constructed using flip-flop circuits, allowing for swift and random access by the CPU.
- RAM is categorized into two main types:
    1. **Static RAM (SRAM):** SRAM is a type of RAM that uses flip-flop circuits to store each bit of data. It is faster and more expensive than Dynamic RAM (DRAM). SRAM retains its data as long as power is supplied, making it suitable for cache memory.
        - In static RAM (SRAM), each memory cell is typically constructed using 6 transistors and does not involve the use of capacitors. The absence of capacitors sets

SRAM apart from dynamic RAM (DRAM). Instead of capacitors, SRAM utilizes flip-flops to store data, eliminating the need for constant refreshing.
    - One notable advantage of SRAM is that it does not require the frequent refreshing that is characteristic of DRAM. The use of flip-flops for data storage allows SRAM to retain its contents without the need for continuous refreshing. However, it's essential to note that data stored in SRAM is volatile and will be lost when the power supply is disrupted or turned off.
2. **Dynamic RAM (DRAM):** DRAM, on the other hand, requires constant refreshing of the stored data as it tends to leak away over time. It is slower compared to SRAM but is more cost-effective and provides higher storage capacity. DRAM is commonly used as the main memory in computer systems.
    - In dynamic RAM (DRAM), each memory cell is constructed using a combination of one transistor and one capacitor. The unique design of DRAM allows it to store data in the form of an electric charge within the capacitor. However, there is a notable characteristic of DRAM cells – the capacitor tends to discharge relatively quickly.
    - Due to this inherent property, dynamic RAM requires constant refreshing to maintain the integrity of stored data. The process involves charging the capacitor thousands of times per second to compensate for the discharge and ensure that the data remains intact.
- RAM plays a crucial role in the overall performance of a computer, providing quick access to data that is actively being used by the CPU. The distinction between SRAM and DRAM highlights the trade-offs between speed, cost, and capacity in different computing scenarios.

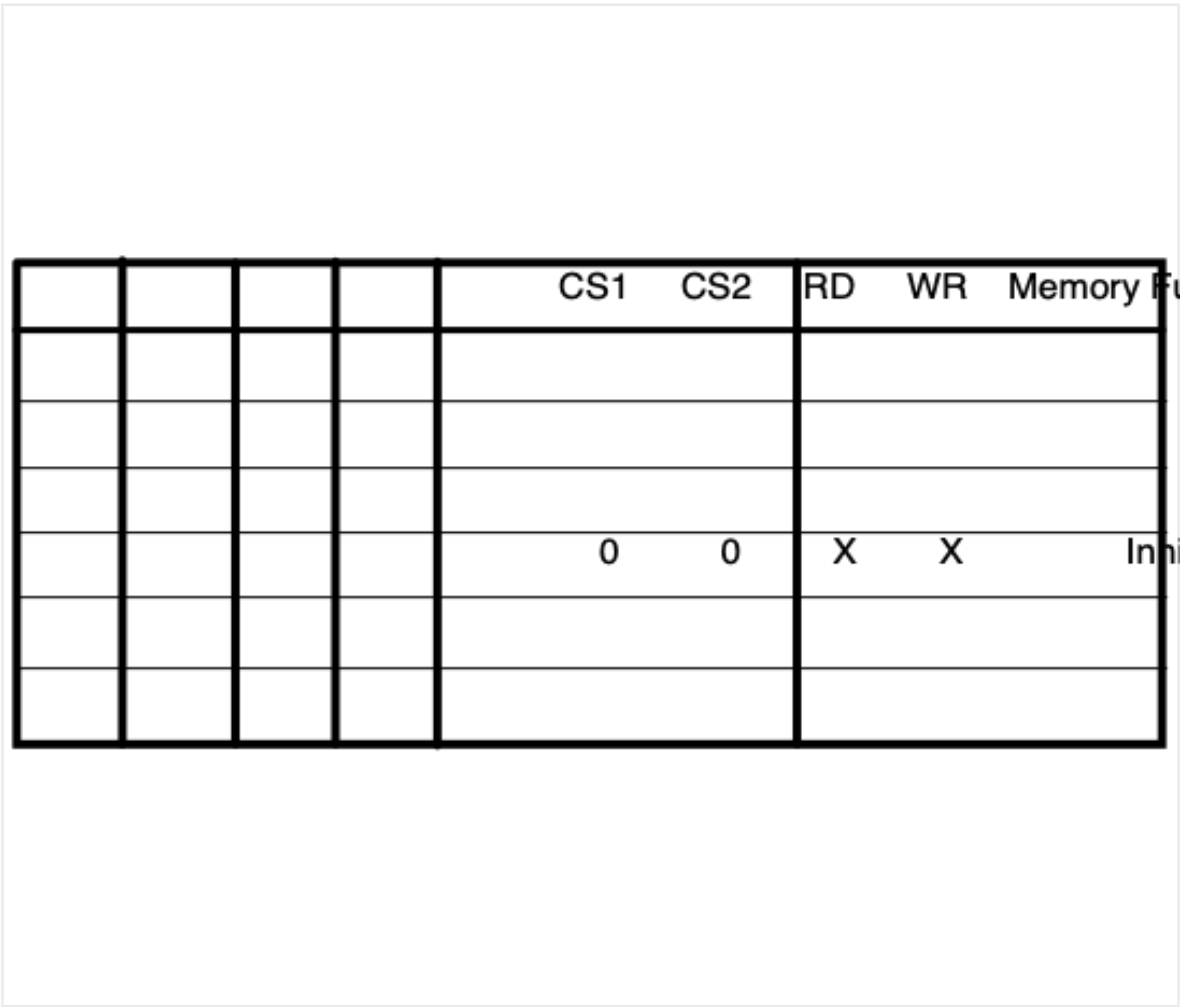
RAM Block Diagram

The block diagram below illustrates the key components of a Random Access Memory (RAM).



- The RAM has a capacity of 128 words, with each word consisting of eight bits. This configuration necessitates a 7-bit address for memory addressing. The 8-bit bidirectional data bus facilitates the transfer of data to and from the memory.
- The memory's read and write operations are controlled by specific lines. The read line is utilized for fetching data from the RAM, while the write line is responsible for storing data into the memory.
- Two control lines, CS1 and CS2, are employed to activate the RAM chip. It's important to note that the RAM becomes active when CS1 equals 1 and CS2 equals 0. These control lines play a crucial role in managing the access and functionality of the memory.
- In summary, the RAM block diagram demonstrates the interplay of various components, including the address lines, data bus, read and write control lines, and chip select lines, to enable efficient memory operations.

The function table



				CS1	CS2	RD	WR	Memory F
				0	0	X	X	In

- Case 1 : When CS1 = 0 and CS2 = 0. In this condition RAM is not activated. So reading and writing is not possible.
- Case 2 : When CS1 = 0 and CS2 = 1. In this condition RAM is not activated. So reading and writing is not possible.
- Case 3 : When CS1 = 1 and CS2 = 0. In this condition RAM is activated. But RD = 0 and WR = 0, so reading and writing is not possible.
- Case 4 : When CS1 = 1 and CS2 = 0. In this condition RAM is activated. Now RD = 0 and WR = 1, so we can write (store) data into RAM.
- Case 5 : When CS1 = 1 and CS2 = 0. In this condition RAM is activated. Now RD = 1 and WR = 0, so we can read (fetch) data from RAM.
- Case 6 : When CS1 = 1 and CS2 = 1. In this condition RAM is not activated. So reading and writing is not possible.

Read-Only Memory (ROM)

- ROM, an acronym for Read-Only Memory, is a type of non-volatile memory.
- Unlike Random Access Memory (RAM), ROM is designed for

read-only operations, meaning that data stored in ROM can only be read and not modified.
- It serves as a permanent storage medium, and the data within a basic ROM cannot be altered or written to after manufacturing.
- Being non-volatile implies that the information stored in ROM is retained even when the power is turned off.
- During the manufacturing process, information is permanently written into ROM, making it an ideal choice for storing critical system instructions and data that should remain unchanged.
- ROM includes a specialized program known as a bootstrap loader, which plays a crucial role in initiating the startup process of a computer. The bootstrap loader is essential for loading the operating system into the computer's memory.
- While commonly associated with computers, ROM chips find application in various electronic devices such as washing machines and microwave ovens, where the need for permanent and unalterable data storage arises.

Types of ROM

- **PROM (Programmable Read-Only Memory):**
  - PROM is a type of read-only memory that allows users to program it once with desired data.
  - Users purchase a blank PROM and input the required data. Once programmed, the data becomes permanent and cannot be modified.
  - Programming PROM involves burning small fuses within the chip, making it a one-time programmable and non-erasable memory.
  - PROM is suitable for applications where the data is fixed and does not need frequent updates.
- **EPROM (Erasable and Programmable Read-Only Memory):**
  - EPROM can be erased and reprogrammed using special electrical signals or ultraviolet (UV) rays.
  - EPROMs that employ UV rays for erasure are referred to as UVEPROM, while those using electrical signals are known as EEPROM (Electrically Erasable Programmable Read-Only Memory).
  - During programming, an electrical charge is stored in EPROM, and this charge is retained for more than 10

years, providing non-volatile memory storage.
    - EPROM offers the advantage of reusability, making it suitable for applications where data updates are required, but not as frequently as EEPROM.
- **EEPROM (Electrically Erasable Programmable Read-Only Memory):**
    - EEPROM is programmed and erased electrically, allowing for greater flexibility compared to EPROM.
    - It supports erasing and reprogramming cycles, typically up to ten thousand times, making it more versatile for applications requiring frequent updates.
    - Both erasing and programming in EEPROM are relatively quick, taking about 4 to 10 milliseconds (ms).
    - EEPROM provides the capability to selectively erase and program specific memory locations, offering a granular approach to data modification.
    - Applications for EEPROM include settings storage in electronic devices, where frequent updates or customization of data are necessary.

Cache Memory

- The main memory's speed is significantly lower compared to the speed of processors, impacting overall system performance.
- To address this, a fast cache memory is employed to reduce memory access time, ensuring that the processor can retrieve data quickly without spending excessive time accessing the main memory.
- The efficiency of the cache mechanism relies on the principle of "locality of reference," where a set of data or instructions is accessed repeatedly.
- Cache memory, typically constructed using fast Static Random Access Memories (SRAMs), stores frequently accessed data or instructions.
- When the processor needs data or instructions, it first checks the cache. If the required information is found in the cache, a "Hit" occurs, avoiding the need to access the slower main memory. If the information is not in the cache, the processor proceeds to access the main memory and, if necessary, the secondary memory.
- The access time of cache memory is faster than that of the

main memory, contributing to improved system performance.

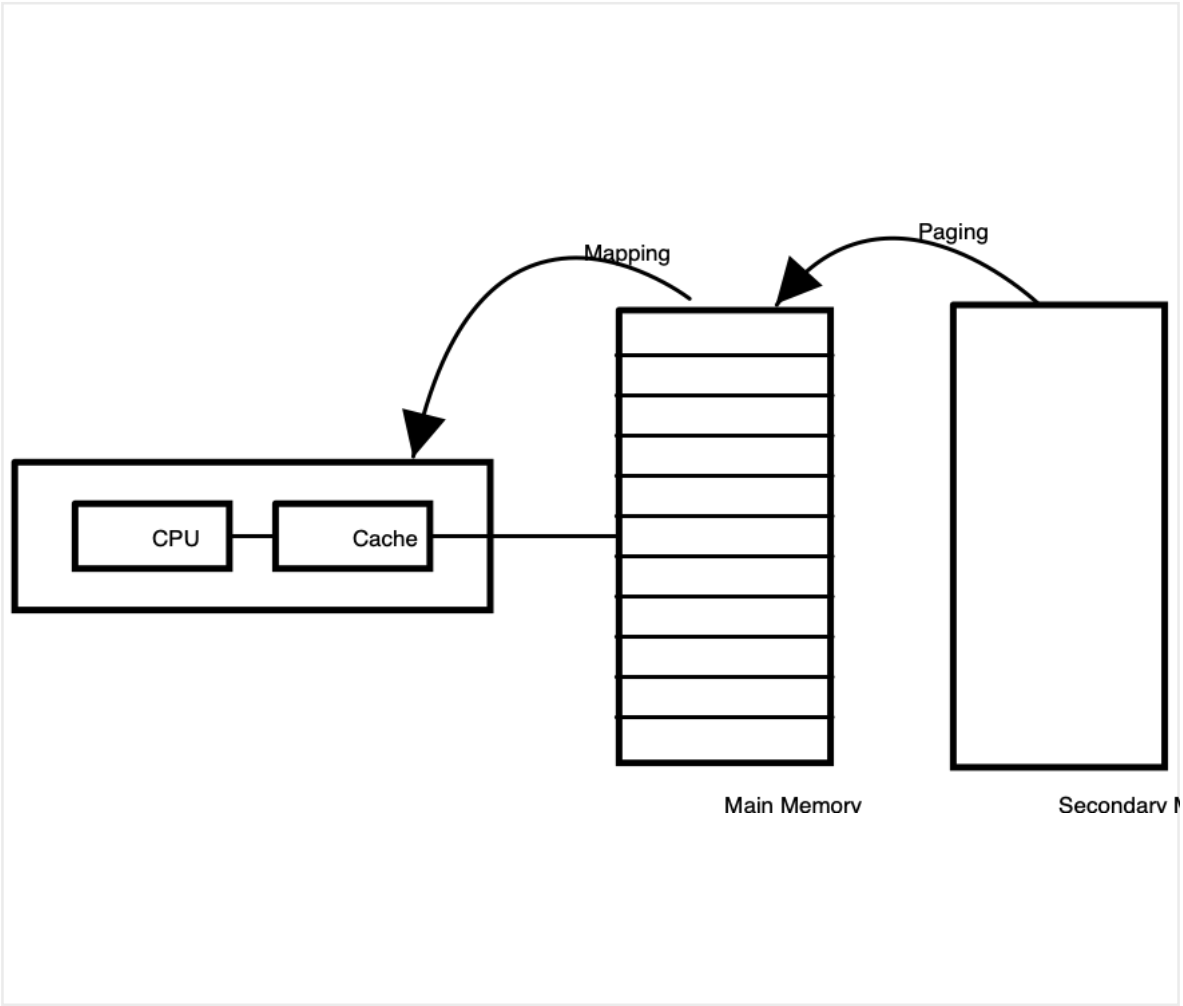
- The performance of cache memory is quantified by a metric known as the "hit ratio," which is the ratio of the number of hits to the total number of memory accesses (hits and misses).
- Mathematically, the Hit Ratio (H) is calculated as: H = Number of Hits / (Number of Hits + Number of Misses).

Types of Cache Memory

- **Level 1 Cache (L1 Cache):**
  - L1 Cache is the primary cache located on the processor chip itself.
  - It is designed to store a small amount of frequently accessed data and instructions for quick retrieval.
  - Due to its proximity to the processor, L1 Cache offers extremely fast access times, contributing to enhanced overall system performance.
  - L1 Cache is further divided into separate caches for instructions (L1i) and data (L1d).
- **Level 2 Cache (L2 Cache):**
  - L2 Cache is located on a separate chip but is still situated near the processor.
  - It has a larger capacity compared to L1 Cache and serves as a secondary cache layer.
  - L2 Cache helps bridge the speed gap between the processor and the main memory.
  - Both instructions and data are stored in the unified L2 Cache, making it a crucial component for optimizing system performance.
- **Level 3 Cache (L3 Cache):**
  - L3 Cache is a shared cache that may be located on the processor chip or a separate chip.
  - It has an even larger capacity compared to L2 Cache and is shared among multiple processor cores within a system.
  - L3 Cache helps improve the overall efficiency of multi-core processors by providing a larger pool of shared cache memory.
  - Its larger size and shared nature make L3 Cache effective in handling diverse workloads and improving system-level performance.

- **Unified Cache:**
  - Unified Cache combines the storage of instructions and data in a single cache, unlike separate L1i and L1d caches.
  - It simplifies cache management and reduces the complexity of addressing both instruction and data caches separately.
  - Unified Caches are commonly found in modern processors to streamline memory access and enhance overall efficiency.

Cache Mapping



- Cache mapping is a crucial technique employed to manage the transfer of content from the main memory to the cache memory.
- It involves the transformation of data from the comparatively slower main memory to the faster cache memory, optimizing the overall speed of data access for the processor.
- In the architecture, the CPU is directly connected to the cache memory, forming a high-speed link crucial for rapid data retrieval.

- The cache memory, in turn, is connected to the main memory, which serves as an intermediary between the fast cache memory and the slower secondary memory.
- The primary purpose of cache mapping is to ensure that the memory speed matches the CPU's processing speed. By directly accessing the cache memory, the CPU minimizes the time spent waiting for data retrieval.
- During the mapping process, data is transferred from the main memory to the cache memory, facilitating quick access to frequently used instructions and data by the processor.
- Another related term is "paging," which involves the transfer of data from the secondary memory to the main memory. This process complements cache mapping, ensuring that a hierarchy of memory levels is efficiently utilized to meet the varying speed requirements of different memory types.
- 3 Types of Mapping:
    1. Associative Mapping:
        - Associative mapping is a cache mapping technique where a block of main memory can be placed in any cache location.
        - There is no fixed relationship between the block's address in main memory and its location in the cache.
        - This flexibility allows for efficient utilization of cache space but comes at the cost of increased complexity and hardware requirements.
        - Associative mapping is well-suited for scenarios where the program's memory access patterns are unpredictable, providing high flexibility in cache usage.
    2. Direct Mapping:
        - Direct mapping is a simpler cache mapping technique where each block of main memory is mapped to a specific location in the cache.
        - There is a fixed relationship between the block's address in main memory and its location in the cache, determined by a mathematical function.
        - This method is efficient in terms of hardware complexity but may lead to cache conflicts where multiple blocks contend for the same cache location.
        - Direct mapping is suitable for scenarios where memory access patterns are somewhat predictable and can

benefit from a straightforward mapping approach.

3. Set-Associative Mapping:
    - Set-associative mapping combines features of both associative and direct mapping, providing a compromise between flexibility and simplicity.
    - The cache is divided into sets, and each set contains multiple lines. Each block from main memory can be placed in any line within a specific set.
    - This allows for a degree of flexibility in cache usage while addressing some of the issues associated with direct mapping, such as cache conflicts.
    - Set-associative mapping strikes a balance, making it suitable for a wide range of applications with varying memory access patterns.

Locality of References

- **Definition:** Locality of reference refers to a phenomenon in which a computer program accesses the same set of memory locations very frequently for a particular time period. It signifies the tendency of a computer program to access instructions whose addresses are near one another. This property is often observed in loops within a program.

Types of Localities:

- **Temporal Locality:**
    - Temporal locality implies that current data being fetched may be needed again soon. To exploit this, the data or instruction is stored in the cache memory, eliminating the need to search the main memory for the same data repeatedly.
    - When the CPU fetches data from RAM, it is also stored in the cache memory based on the assumption that the same data or instruction may be needed in the near future. This phenomenon is known as temporal locality.
    - If certain data is referenced, there is a high probability that it will be referenced again in the near future, making temporal locality a key optimization factor in caching.
- **Spatial Locality:**
    - Spatial locality assumes that if a memory location has been accessed, there is a high likelihood that a nearby or consecutive memory location will be accessed soon after. To capitalize on this, nearby memory references are also

stored in the cache memory for faster access.
    - For instance, the traversal of a one-dimensional array in any instruction set benefits from spatial locality optimization.
    - Spatial locality enhances caching efficiency by anticipating and preloading adjacent memory locations that are likely to be accessed shortly.

Virtual Memory

- Virtual memory creates the illusion of a larger memory space, providing the appearance of abundant memory despite physical limitations.
- The technique of virtual memory allows users to utilize more memory for a program than the actual physical memory capacity of a computer.
- It is a concept that conveys the illusion to users that their available main memory is equal to the capacity of secondary storage media (or auxiliary memory).
- **Need for Virtual Memory:**
    1. Virtual memory is an imaginary memory concept used when a program exceeds the size of the available main memory.
    2. It functions as temporary memory, complementing the RAM (or main memory) of the system.
- Virtual Memory (VM) is a memory management capability of an operating system that utilizes both hardware and software to address physical memory shortages by temporarily transferring data between RAM and disk storage.
- **Example Scenario:** Consider a system with 4 GB of main memory and a program size of 16 GB. Since it is not feasible to store the entire 16 GB program in the 4 GB main memory at once, virtual memory comes into play. A portion of the 16 GB program that is currently in use is transferred to the main memory. When that portion is no longer needed, it is moved back to the secondary memory in a process known as swapping.
- The fundamental idea behind virtual memory is to keep only those parts of the program currently in use in the memory, while the rest remains on the disk drive.

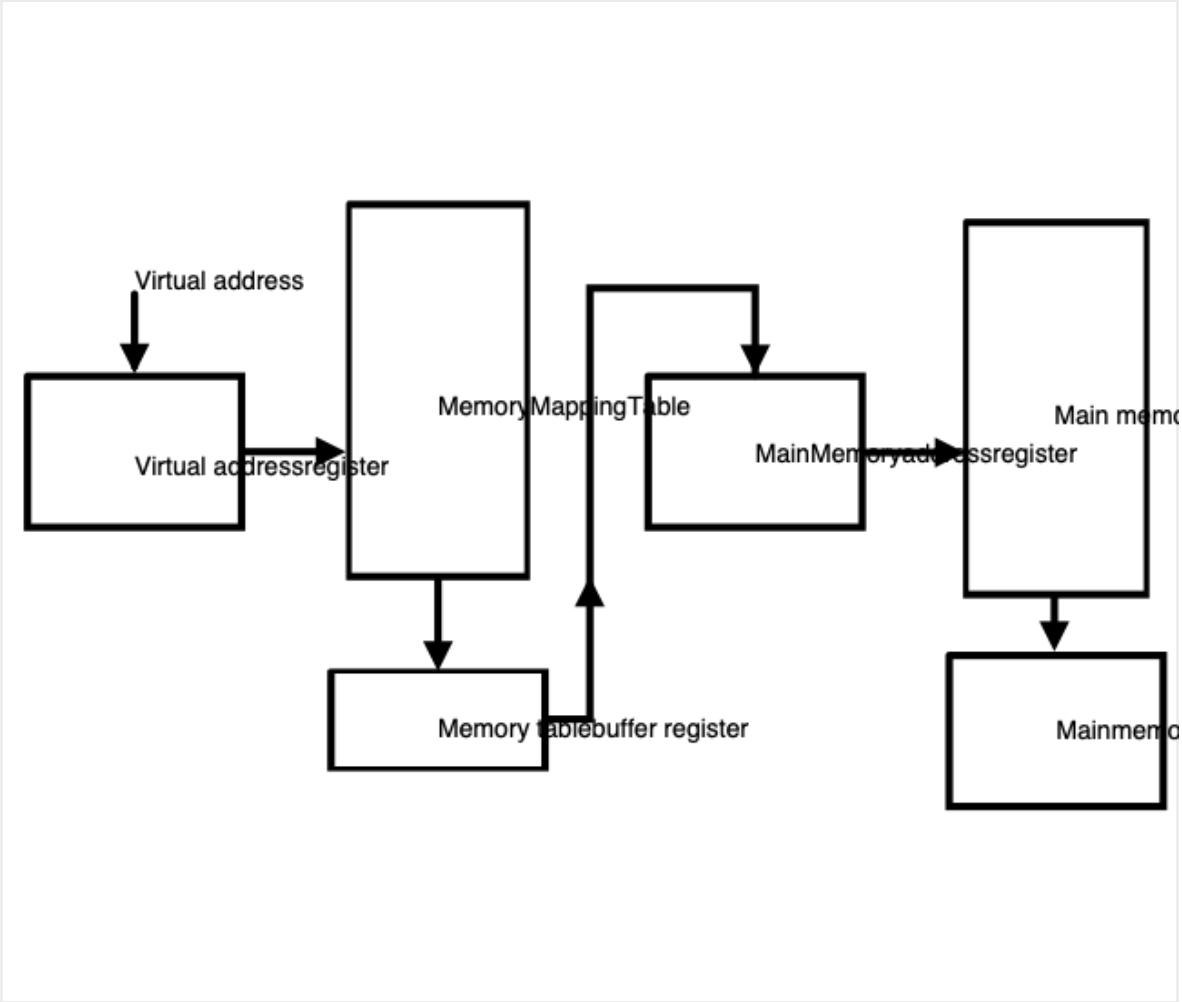
Address Space and Memory Space

- **Virtual Address and Address Space:**

    - An address used by a programmer is referred to as a virtual address, and the collective set of these addresses constitutes the address space.
    - The address space represents the range of virtual addresses that a program can use during its execution.
    - It provides an abstraction for the programmer, offering a seemingly continuous and expansive range of addresses, regardless of the underlying physical memory layout.
- **Physical Address and Memory Space:**
    - An address in the main memory, corresponding to a location where data or instructions are stored, is termed a physical address. The set of these addresses constitutes the memory space.
    - Memory space represents the actual locations within the physical memory hardware where data is stored and can be directly accessed.
    - It reflects the real and finite capacity of the physical memory modules installed in the computer system.
- Each address that is referenced by the CPU goes through an address mapping (or address translation) from so called virtual address to a physical address in main memory.
- Virtual memory system provides a mechanism for transalting program-generated addresses into correct main memory locations dynamically.
- The address translation or mapping is handled automatically by the hardware by means of a mapping table.

Memory Table for Mapping a Virtual Address

- Virtual addresses, typically processed through memory mapping tables, are translated to physical addresses or mapped. The process can be better understood with the help of the following steps:



- First, the virtual address is received in the virtual address register.
- Subsequently, the virtual address is sent to the memory mapping table, which holds crucial information about the location of the address in the main memory and its accessibility.
- The details, including the location of the address in the main memory, are stored in the memory table buffer register.
- This information is then transferred to the main memory address register, now represented as a physical address, where it is stored.
- With the physical address, the data in the main memory can be accessed.
- The accessed data is stored in the main memory buffer register.

There are two primary virtual memory management techniques that handle the mapping from virtual addresses to physical addresses:

1. **Paging (Specifically Known as Demand Paging)**

Paging is a virtual memory management technique that divides both virtual and physical memory into fixed-sized pages. When a process needs data from the disk, only the necessary pages are loaded into the main memory. Demand Paging optimizes memory usage by bringing in data on-demand, reducing the initial load time and allowing for more efficient utilization of resources.

2 **Segmentation (Demand Segmentation)**

Segmentation is another virtual memory management approach that divides memory into variable-sized segments. Each segment represents a logical unit, such as a function or a data structure. When a process requires a specific segment, only that segment is loaded into memory. Demand Segmentation allows for flexibility in managing memory spaces and is particularly useful in scenarios where the size of data structures varies dynamically.