# BCSE205L
# Computer Architecture and Organization

## Module 5 – Interfacing and Communication

| Module:5 | Interfacing and Communication | 5 Hours |
|---|---|---|
| I/O fundamentals: handshaking, buffering, I/O Modules - I/O techniques: Programmed I/O, Interrupt-driven I/O, Direct Memory Access, Direct Cache Access - Interrupt structures: Vectored and Prioritized-interrupt overhead - Buses: Synchronous and asynchronous - Arbitration. | | |

**Dr. C.R.Dhivyaa**

**Assistant Professor**

**School of Computer Science and Engineering**

**Vellore Institute of Technology, Vellore**

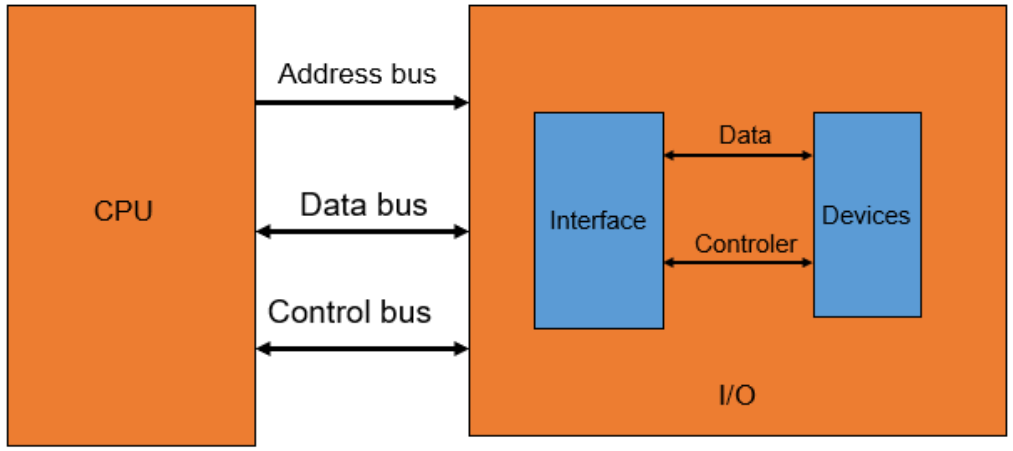
# Introduction to I/O Module

- I/O is referred as Input-Output of a computer
- I/O devices are connected to CPU through I/O module called as Interfacing
- What for I/O?
  - I/O module provides an efficient mode of communication between CPU and outside of environment
- Why to use I/O module? Why can't connect peripheral devices directly to the subsystem?
  - Diversity
  - Speed
  - Control

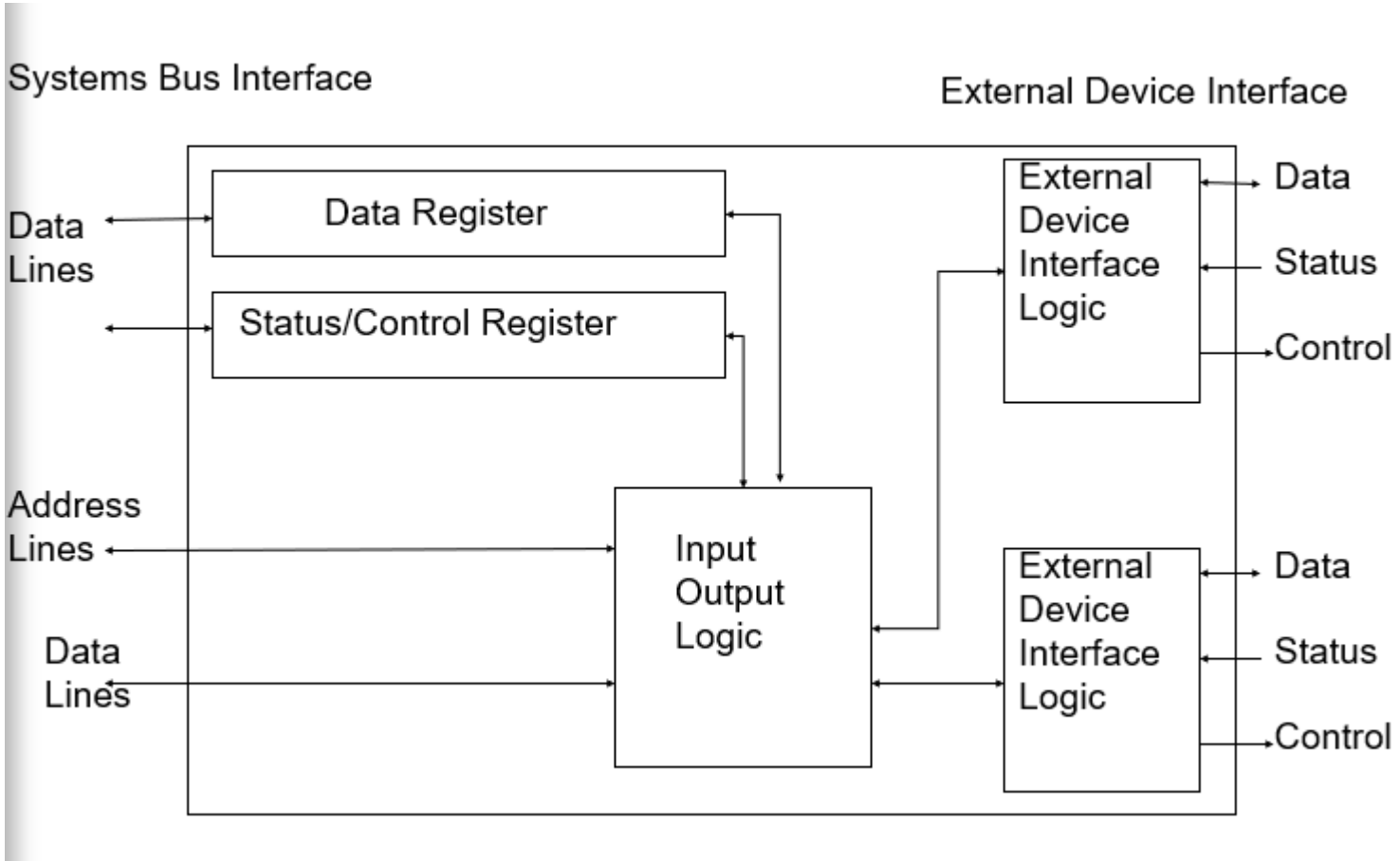
# Functions of I/ O Module

- **Processor communication**
  - Data transfer between the processor and an I/O module
  - Accepting and decoding commands sent by the processor
  - Reporting of current status
- **Device communication** – It needs to be able to perform standard device communications, such as reporting of status.
- **Control and timing** – An I/O module needs to be capable of managing data flow between a computer's internal resources and any connected external devices.
- **Data buffering** – A crucial function that manages the speed discrepancy that exists between the speed of transfer of data between the processor and memory and peripheral devices.
- **Error detection** – Detecting errors, whether mechanical (such as a printer experiencing a paper jam) or data based, and reporting them to the processor

# CPU & I/O Module Organization




- Many I/O devices
- Each device - unique address
- Processor- Command- address of IO device
- Processor, Main memory, IO - Share a common bus



# Data Transfer

- **DATA TRANSFER**
  - The physical transfer of data (a digital bit stream) over a point-to-point or point-to-multipoint communication channel.
  - Example (channels): copper wires, optical fibres, wireless communication channels.
- **SYNCHRONOUS DATA TRANSFER**
  - The transfer of data between two devices on a network where they both carry out a predetermined set of interactions based on a common clock pulse.
- **ASYNCHRONOUS DATA TRANSFER**
  - The transfer of data between two devices on a network where they both carry out a predetermined set of interactions based on a private clock pulse.

# Synchronous Data Transfers

- If the registers in the I/O interface share a common clock with CPU registers, then transfer between the two units is said to be **Synchronous**
- It usually occur when peripherals are located within the same computers as the CPU
- It shares a common clock

# Asynchronous Data Transfers

- Asynchronous data transfer is a method of data transmission where data is sent in a non-continuous, non-synchronous manner. This means that data is sent at irregular intervals, without any specific timing or synchronization between the sending and receiving devices.
- Asynchronous Data Transfer between two independent units requires that control signals be transmitted between the communicating units so that the time can be indicated at which they send data.
- **Control signals**
  - **Strobe control:** A strobe pulse is supplied by one unit to indicate to the other unit when the transfer has to occur.
  - **Handshaking:** This method is commonly used to accompany each data item being transferred with a control signal that indicates data in the bus. The unit receiving the data item responds with another signal to acknowledge receipt of the data.

# Asynchronous Data Transfer Methods

## Methods to accomplish Asynchronous Data Transfer

i. Source Initiated Strobe for Data Transfer.

ii. Destination Initiated Strobe for Data Transfer.

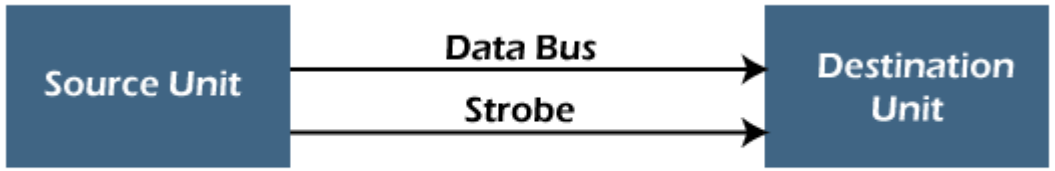
iii. Source Initiated Transfer Using Handshaking.

iv. Destination Initiated Transfer Using Handshaking.

# Asynchronous Data Transfer Methods

## i) Source Initiated Strobe for Data Transfer

- Source unit places data on the data bus and activates Strobe pulse.
- Data bus carries binary information from source to destination.
- Information on Data bus remains in Active state till the data is received by destination.
- Destination unit uses falling edge to transfer data.
- Source removes data from the bus and disables the Strobe pulse.



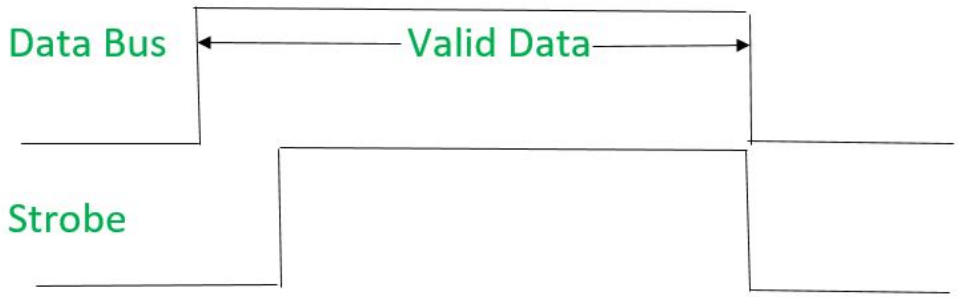

(a) Block Diagram

(i) First, source puts data on the data bus and ON the strobe signal

(ii) Destination on seeing the ON signal of strobe, read data from the data bus.

(iii) After reading data from the data bus by destination, strobe gets OFF.

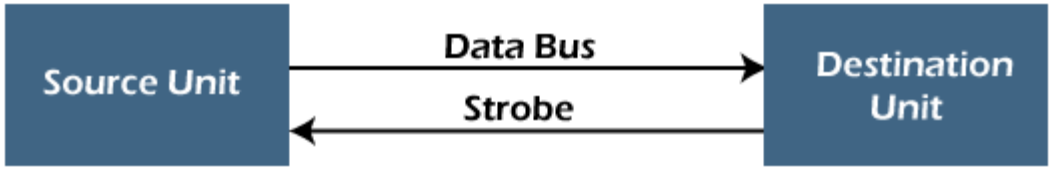
Fig. shows that first data is put on the data bus and then strobe signal gets active.



# Asynchronous Data Transfer Methods

## ii) Destination Initiated Strobe for Data Transfer.

- Destination unit activates Strobe pulse.
- Source responds by placing the requested data on the bus.
- Data must be valid and available for small period of time, so destination can accept the information.
- Destination unit disables the Strobe pulse.
- Source removes the data from the bus.



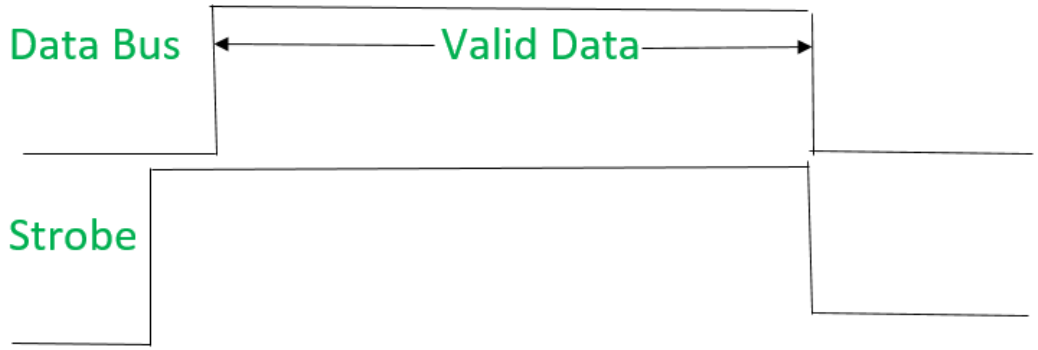

(a) Block Diagram

(i) First, the destination ON the strobe signal to ensure the source to put the fresh data on the data bus.

(ii) Source on seeing the ON signal puts fresh data on the data bus.

(iii) Destination reads the data from the data bus and strobe gets OFF signal.

*Fig. shows that first strobe signal gets active then data is put on the data bus*



# Asynchronous Data Transfer Methods

- **Problems faced in Strobe based asynchronous**
  - **No Acknowledgement for Data Sent/Received.**
  - In Source initiated Strobe, it is assumed that destination has read the data from the data bus but there is no surety.
  - In Destination initiated Strobe, it is assumed that source has put the data on the data bus but there is no surety.
  - This problem is overcome by **Handshaking**

# Asynchronous Data Transfer Methods

## iii)Source Initiated Transfer Using Handshaking

- When source initiates the data transfer process.
- It consists of signals:
  - DATA VALID: if ON tells data on the data bus is valid otherwise invalid.
  - DATA ACCEPTED: if ON tells data is accepted otherwise not accepted.

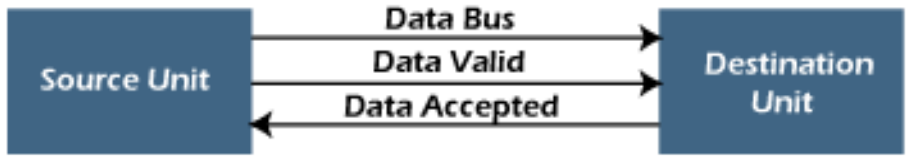
i) Source places data on the data bus and **enable Data valid** signal.

(ii) Destination accepts data from the data bus and **enable Data accepted** signal.

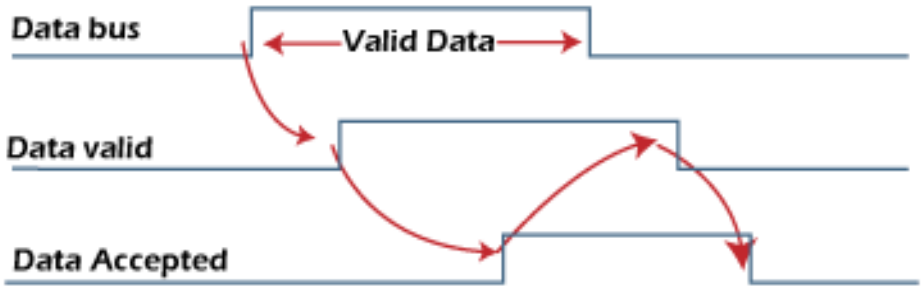
(iii) After this, **disable Data valid signal** - means data on data bus is invalid now.

(iv) **Disable Data accepted signal** and the process ends.

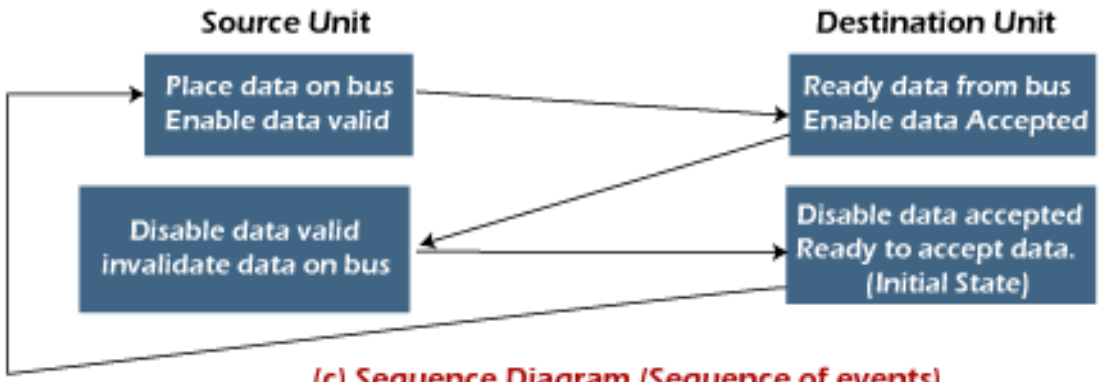
Now there is surety that destination has **read** the data from the data bus through data accepted signal.




(a) Block Diagram




(b) Timing Diagram




(c) Sequence Diagram (Sequence of events)

# Asynchronous Data Transfer Methods

## iv) Destination Initiated Transfer Using Handshaking

- When destination initiates the process of data transfer.
- READY/REQUEST FOR **DATA**: if ON requests for putting data on the data bus.
- DATA VALID: if ON tells data is valid on the data bus otherwise invalid data.

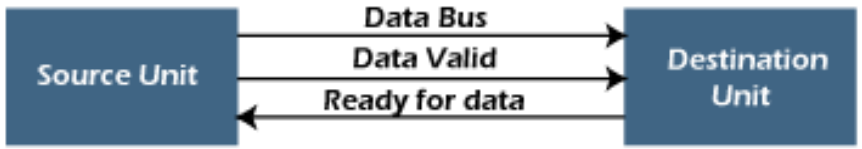
  (i) When destination is ready to receive data, **Ready for Data signal gets activated**.

  (ii) Source in response **puts data on the data bus and enabled Data valid signal**.

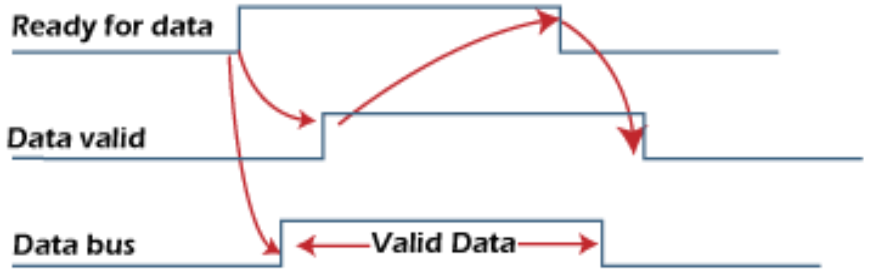
  (iii) Destination then accepts data from the data bus and after accepting data, **disabled Request/Ready for Data signal.**

  (iv) At last, **Data valid signal gets disabled** means data on the data bus is no more valid data.

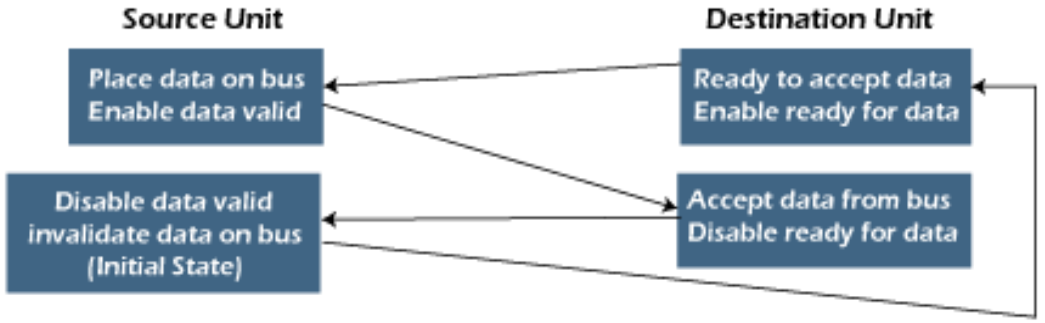
  Now there is surety that source has **put the data** on the data bus through data valid signal.




(a) Block Diagram




(b) Timing Diagram




(c) Sequence Diagram (Sequence of events)

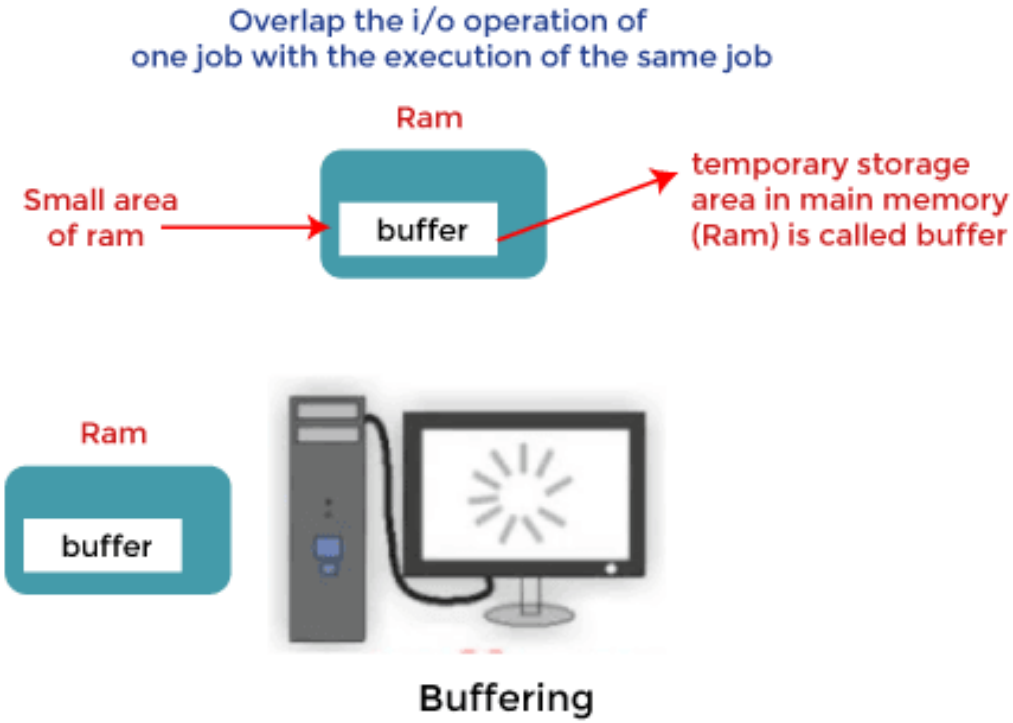
# Asynchronous Data Transfer Methods

- Advantages:
  - Can be used to transfer any kind of data as it is flexible.
    (i.e. Text, Audio, video, etc)
  - Used in web applications like email, chatting.

# Buffering

- Buffering is a technique that improves the efficiency and throughput of I/O operations.
- Plays a very important role in any OS during the execution of any process or task.
- Involves temporarily storing data while it's being transferred between two devices or processes.
- Buffering helps to
  - Synchronize two devices with different processing speeds
  - Help devices with different data transfer sizes adjust to each other
  - Support copy semantics
  - Smooth out peaks in I/O demand

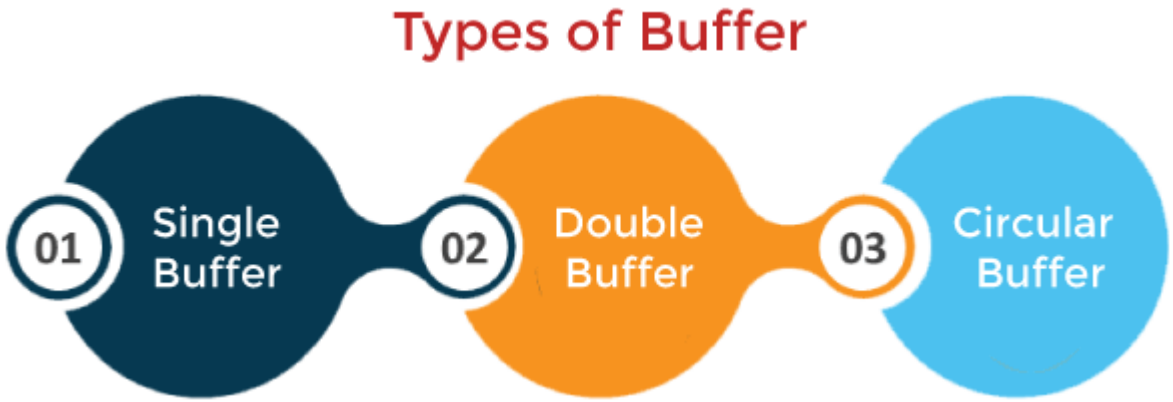
## How Buffering Works?




***Buffers - Implemented using a queue or FIFO algorithm in memory***

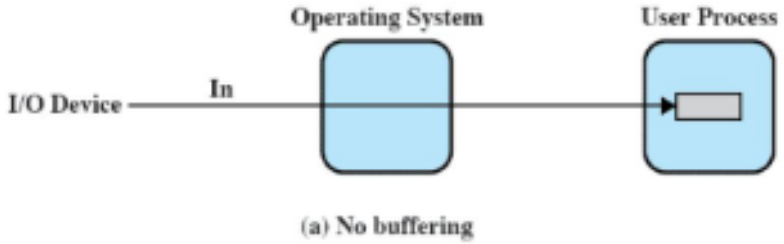
# Types of Buffering

- Single buffering
- Double buffering
- Circular buffering

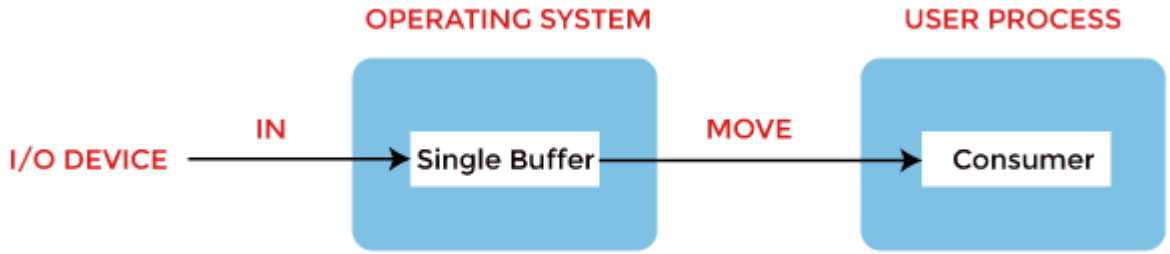


# Types of Buffering

- **Single buffering**
  - In Single Buffering, only one buffer is used to transfer the data between two devices.
  - The producer produces one block of data into the buffer. After that, the consumer consumes the buffer.
  - Only when the buffer is empty, the processor again produces the data.



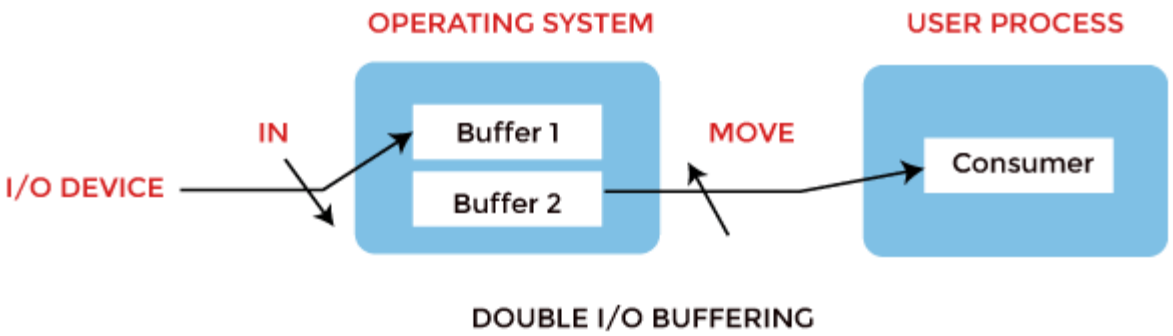
(a) No buffering



SINGLE I/O BUFFERING

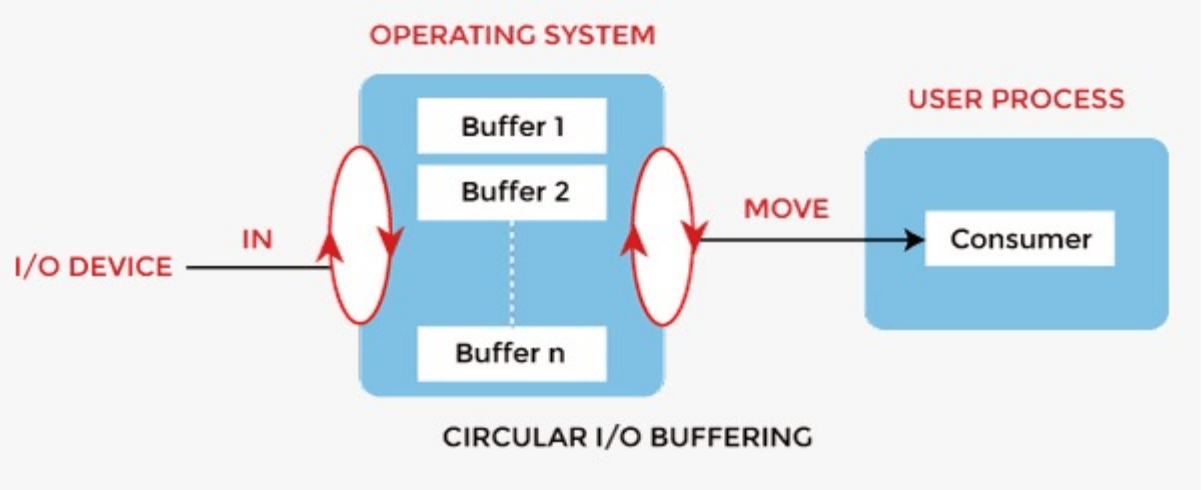
# Types of Buffering

- **Double buffering**
  - Two schemes or two buffers are used in the place of one.
  - Producer produces one buffer while the consumer consumes another buffer simultaneously.
  - So, the producer not needs to wait for filling the buffer.
  - Double buffering is also known as buffer swapping.



# Types of Buffering

- **Circular buffering**
  - More than two buffers are used, the buffers' collection is called a circular buffer.
  - Each buffer is being one unit in the circular buffer.
  - Increases data transfer rate than the double buffering.
  - Data do not directly pass from the producer to the consumer because the data would change due to overwriting of buffers before consumed.
  - The producer can only fill up to buffer x-1 while data in buffer x is waiting to be consumed



# Advantages of Buffering

- Allows uniform disk access
- It simplifies system design
- The system places no data alignment restrictions on user processes doing I/O. By copying data from user buffers to system buffers and vice versa, the kernel eliminates the need for special alignment of user buffers, making user programs simpler and more portable.
- Reduce the amount of disk traffic – increases the overall system throughput and decreases response time.
- The buffer algorithms help ensure file system integrity.

# Disadvantages of Buffering

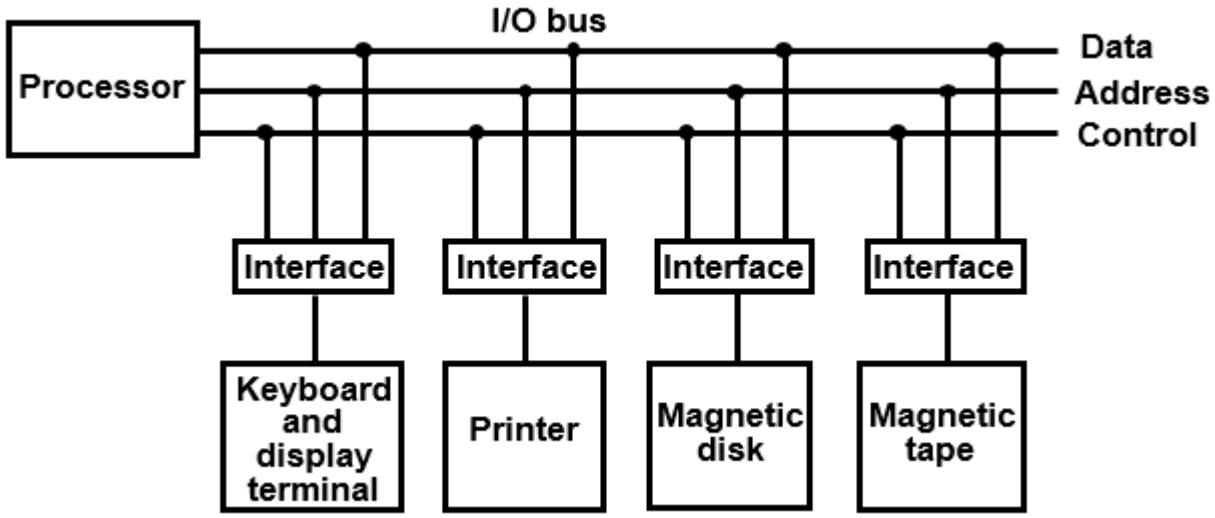
- Costly
- Impractical to have the buffer be the exact size required to hold the number of elements. Thus, the buffer is slightly larger most of the time, with the rest of the space being wasted.
- Buffers have a fixed size at any point in time - When the buffer is full, it must be reallocated with a larger size, and its elements must be moved. Similarly, when the number of valid elements in the buffer is significantly smaller than its size, the buffer must be reallocated with a smaller size and elements be moved to avoid too much waste.
- Use of the buffer requires an extra data copy when reading and writing to and from user processes - when transmitting large amounts of data, the extra copy slows down performance.

# I/O BUS AND INTERFACE MODULES

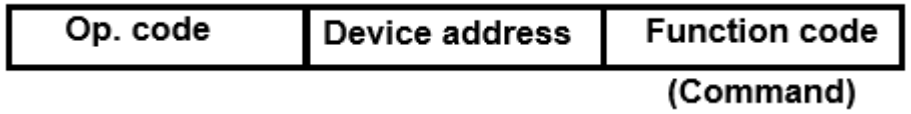
- Each peripheral has an interface module associated with it

## Interface

- Decodes the device address (device code)
- Decodes the commands (operation)
- Provides signals for the peripheral controller
- Synchronizes the data flow and supervises the transfer rate between peripheral and CPU or Memory




### Typical I/O instruction



# I/O techniques

- Data transfer to and from the peripherals may be done in any of the three possible ways
  - Programmed Input/Output
  - Interrupt driven Input/Output
  - Direct Memory Access

| Technique | Interrupt Required | I/O module to/from Memory transfer |
|---|---|---|
| Programmed I/O | No | Through CPU |
| Interrupt driven I/O | Yes | Through CPU |
| Direct Memory Access | Yes | Direct transfer |

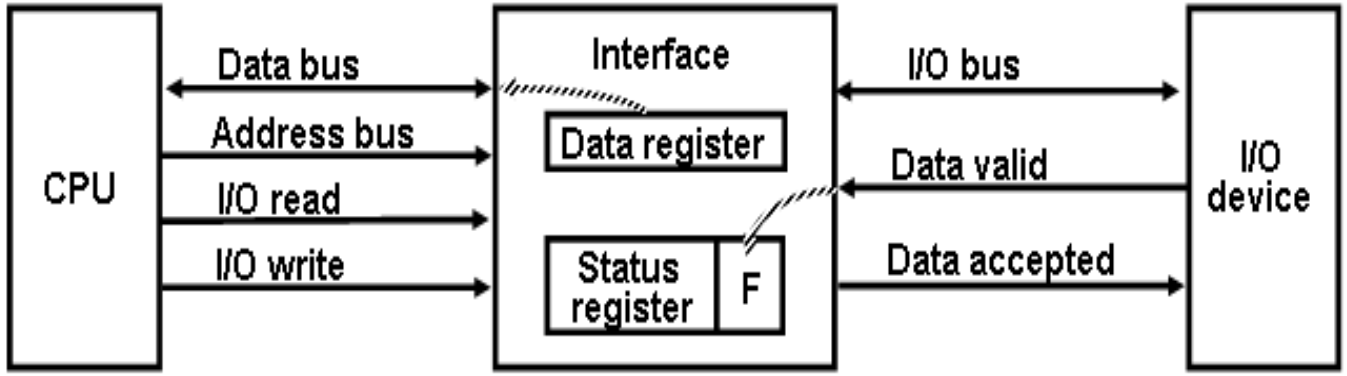
- Interrupt is a response by the processor to an event that needs attention from the software.

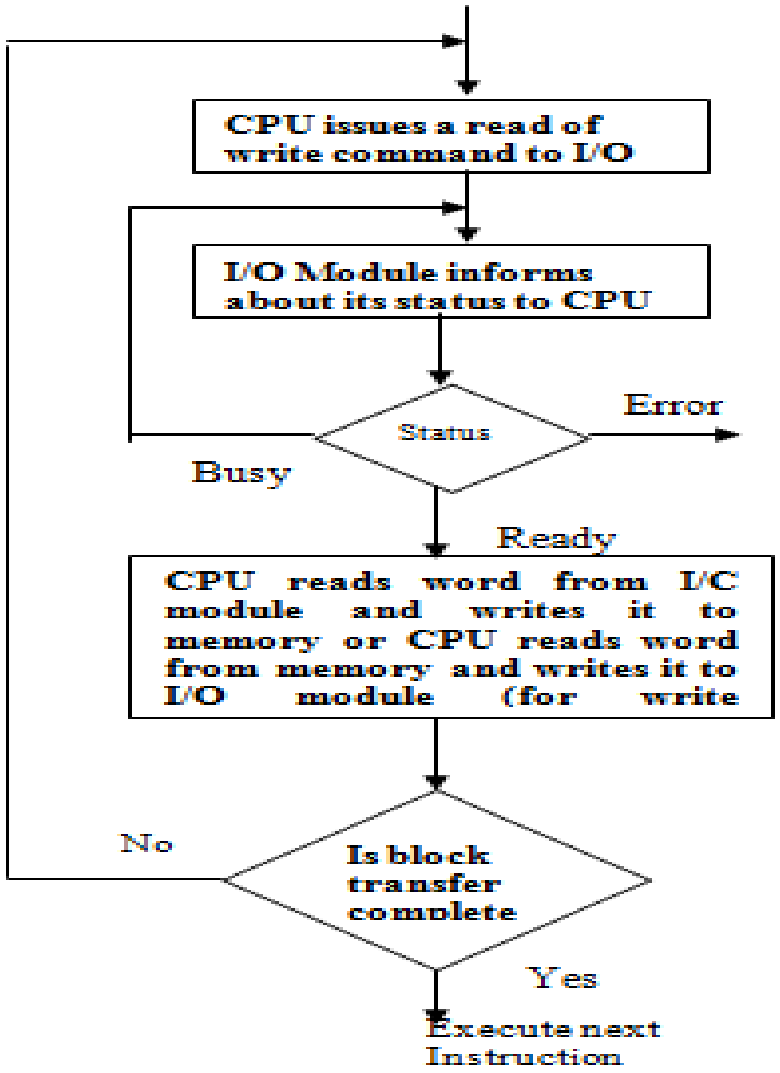
# Programmed I/O

- It is due to the result of the I/O instructions that are written in the computer program.
- Each data item transfer is initiated by an instruction in the program
- Usually the transfer is from a CPU register and memory. Transferring data under this mode requires constant monitoring of the peripheral by CPU.
- Wastes CPU time
- **Example:** A transfer from I/O device to memory requires the execution of several instructions by the CPU involves the following steps.
  - Transfer of data from I/O device to the CPU registers
  - Transfer of data from CPU registers to memory
- **Disadv:** CPU monitors the I/O module and it is always busy - so that the CPU is not free to work on other things – can be avoided using INTERRUPTs

# Programmed I/O - Process

1. CPU requests I/O operation
2. CPU must wait
3. I/O module performs operation
4. I/O module sets status bits
5. CPU checks status bits and then cleans up the operation



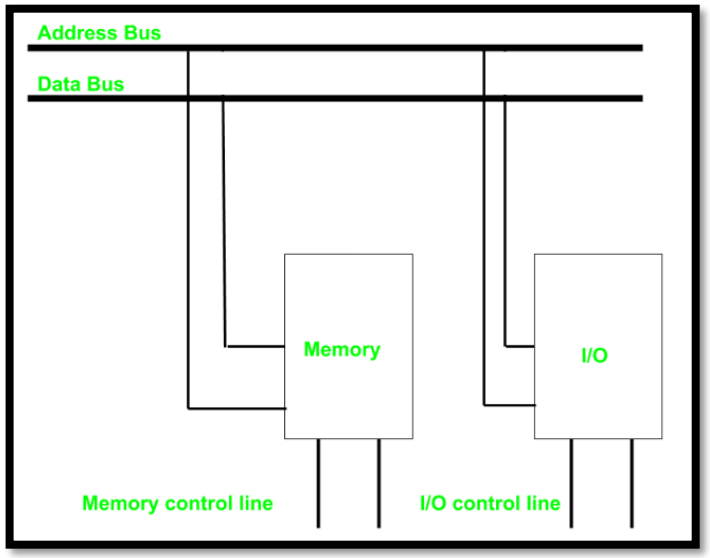



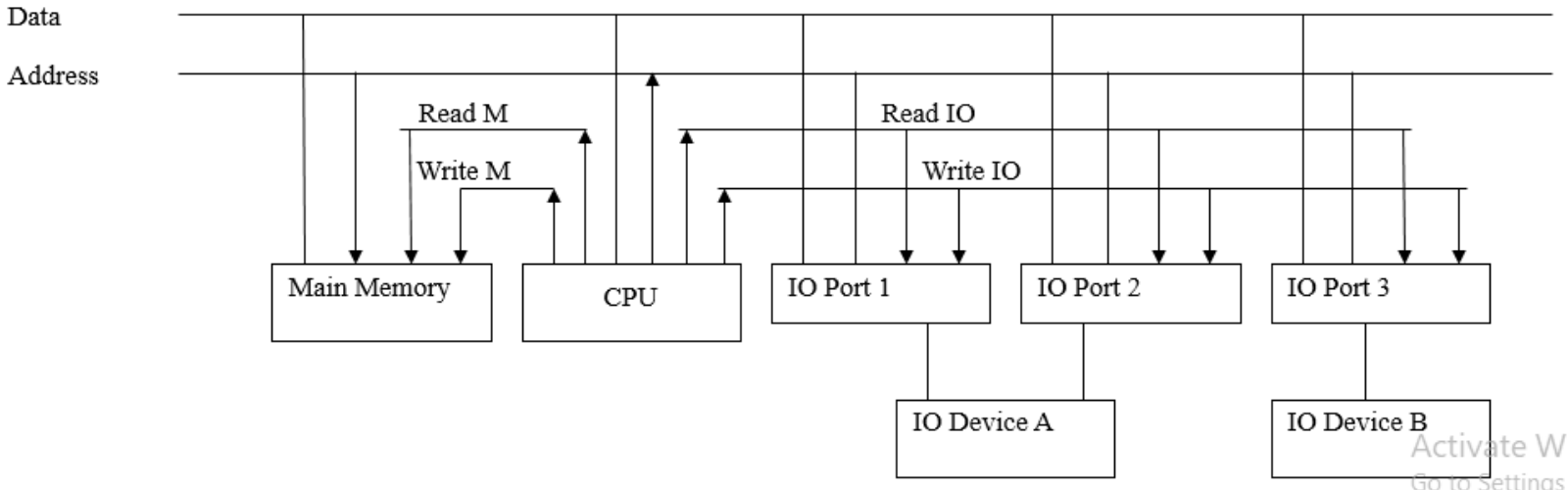
# Programmed I/O – Addressing Methods

- As a CPU needs to communicate with the various memory and input-output devices (I/O) as we know data between the processor and these devices flow with the help of the system bus.
- There are three ways in which system bus can be allotted to them :
  - Separate set of address, control and data bus to I/O and memory.
  - Have common bus (data and address) for I/O and memory but separate control lines **(Isolated I/O)**
  - Have common bus (data, address, and control) for I/O and memory **(Memory Mapped I/O)**

# Programmed I/O – Isolated I/O Addressing

- Common bus(data and address) for I/O and memory
- But separate read and write control lines for I/O
- So when CPU decode instruction, then if data is for I/O then it places the address on the address line and set I/O read or write control line on due to which data transfer occurs between CPU and I/O.
- Address space of memory and I/O is isolated – Isolated I/O.
- The address for I/O - called ports. Different read-write instruction for both I/O and memory.




I/O Mapped I/O

# Programmed I/O – Isolated I/O Addressing

## Advantages of Isolated I/O

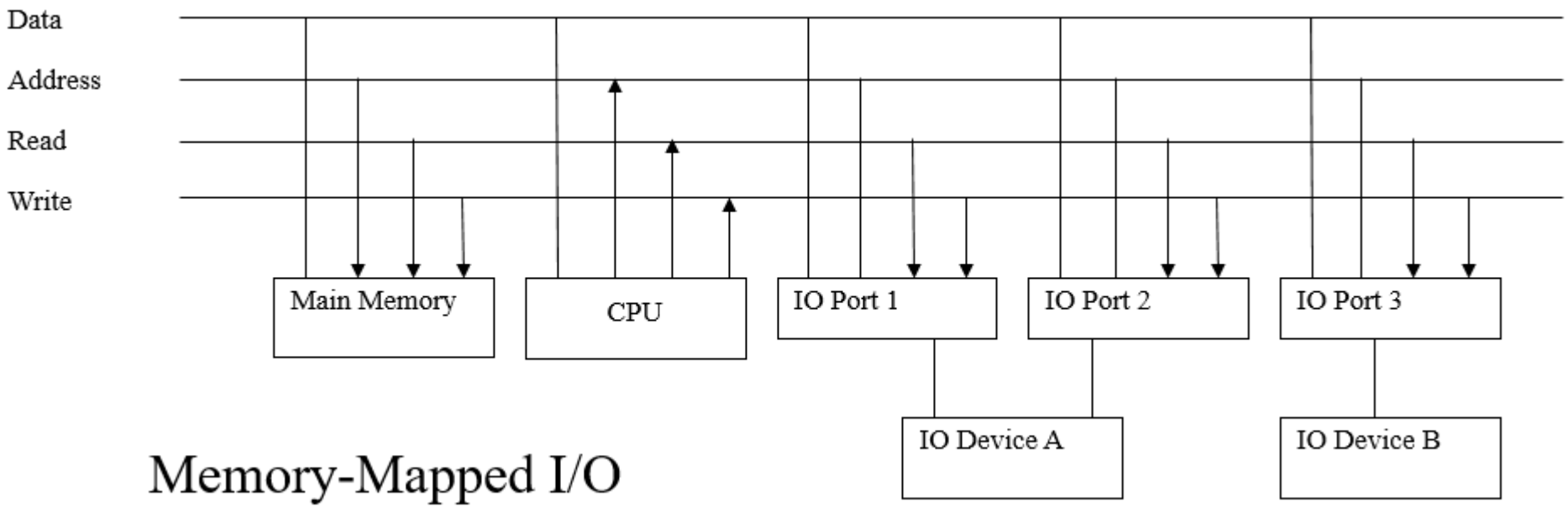
- Large I/O Address Space
- Greater Flexibility
- Improved Reliability

## Disadvantages of Isolated I/O

- Slower I/O Operations
- More Complex Programming

# Programmed I/O - Memory Mapped I/O Addressing

- Common bus (Address, Data, Control Bus) – Memory and I/O.
- I/O and memory - both have same address space
- **Disadv:** Addressing capability of memory become less because some part is occupied by the I/O.


Memory-Mapped I/O

# Programmed I/O - Memory Mapped I/O Addressing

## Advantages of Memory-Mapped I/O

- Faster I/O Operations
- Simplified Programming
- Efficient Use of Memory Space

## Disadvantages of Memory-Mapped I/O

- Limited I/O Address Space
- Slower Response Time

# Isolated I/O & Memory mapped I/O

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

# Programmed I/O

## Advantages

- Simple to implement
- Requires very little special software or hardware

## Disadvantages

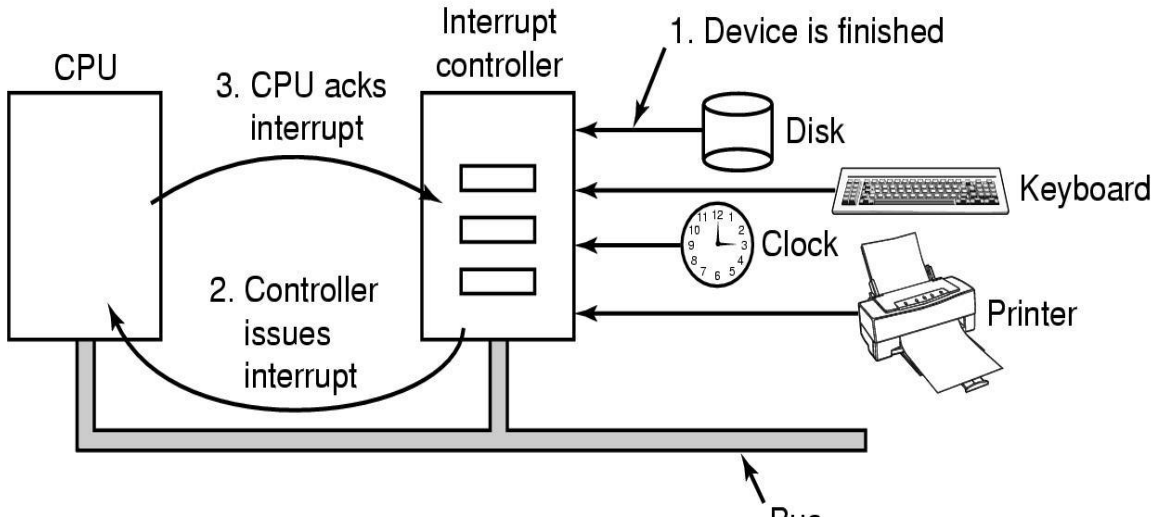
- Speed difference between CPU and the peripheral devices
- programmed I/O wastes an enormous amount of CPU cycles
- Very inefficient
- CPU slowed to the speed of the peripheral

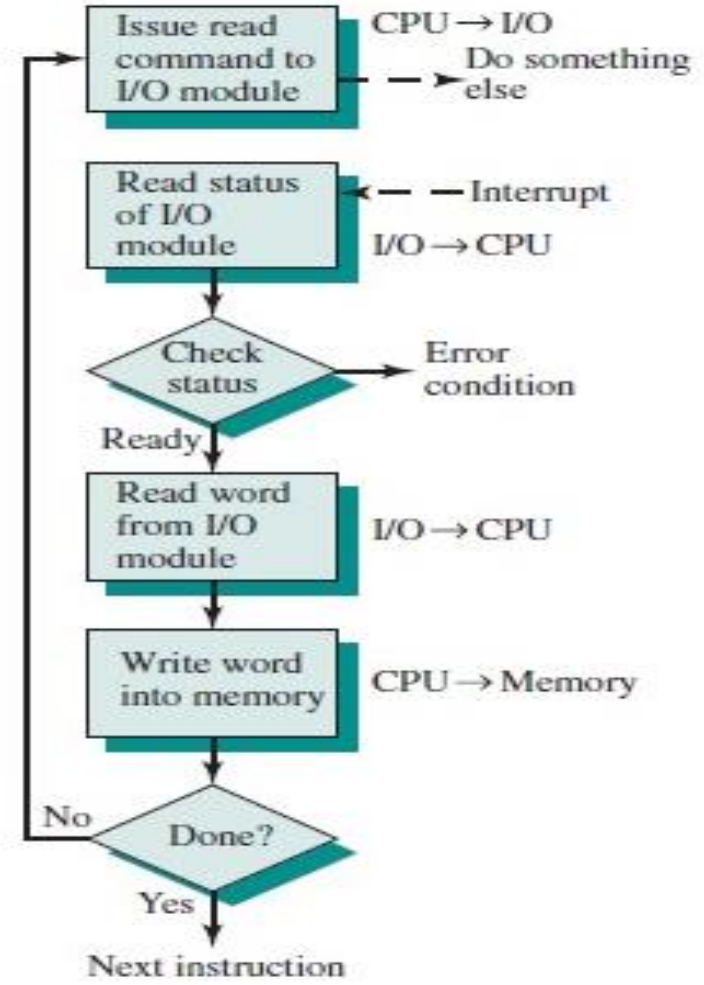
# Interrupt driven I/O

- In Programmed I/O - CPU is kept busy unnecessarily – can be **avoided by using an interrupt** driven method for data transfer.
- By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device (The interface keeps monitoring the device – Not the CPU)
- **CPU → Interface → I/O → Interface → CPU -----→ Memory**
  **(no CPU intervention) (needs CPU intervention)**
- In the meantime the CPU can proceed for any other program execution.
- Whenever it is determined that the device is ready for data transfer **it initiates an interrupt request signal to the computer.**
- Upon detection of an external interrupt signal - CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

# Interrupt driven I/O - Process

- CPU setups and start I/O operation
- CPU goes off to do other things
- When I/O is done, CPU is interrupted
- CPU handles the interrupt
- CPU resumes interrupted operation



# Interrupt driven I/O - Process

- The CPU issues the command to the I/O module
- The CPU then continues with what it was doing
- The I/O module, like before, issues the command to the I/O device and waits for the I/O to complete
- Upon completion of one byte input or output, the I/O module sends an interrupt signal to the CPU
- The CPU finishes what it was doing, then handles the interrupt
- This will involve moving the resulting datum to its proper location on input
- Once done with the interrupt, the CPU resumes execution of the program
  - This is much more efficient than Programmed I/O as the CPU is not waiting during the (time-consuming) I/O process

# Direct Memory Access (DMA)

- Is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, by-passing the CPU to speed up memory operations
- Why DMA?
  - Interrupt driven and programmed I/O require active CPU intervention
  - Transfer rate is limited
  - CPU is tied up
- DMA Function
  - Additional Module (hardware) on bus
  - DMA controller takes over from CPU for I/O transfers
- The process is managed by a chip known as a DMA controller (DMAC)

# Direct Memory Access (DMA)

## Role of CPU in transfer of Information

- Peripheral device -> CPU -> memory (With CPU - Programmed & Interrupted I/O)
  - CPU limits the speed of transfer
- Peripheral device -> memory (Without CPU - DMA)
  - Transfer speed increases
  - Peripheral device manage the memory bus directly.
  - DMA (Direct memory Access)
  - CPU is idle

**Interruped I/O:** **CPU → Interface → I/O → Interface → CPU -----→ Memory**

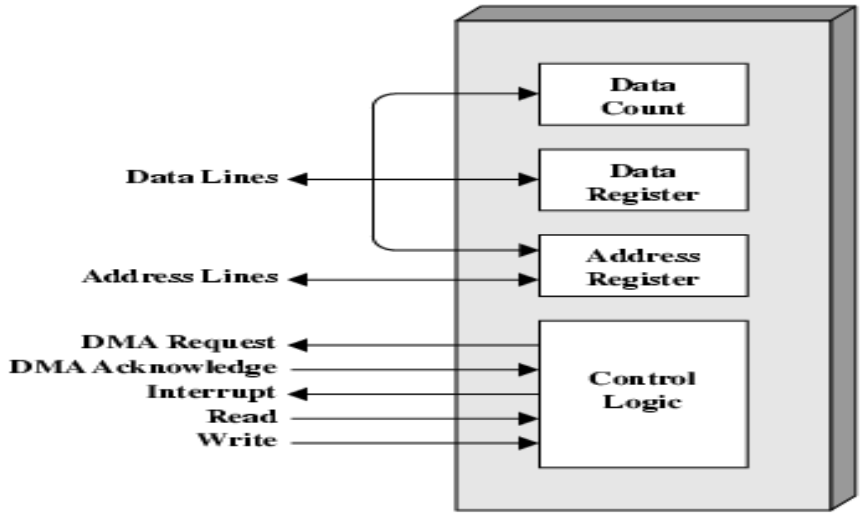
**DMA :** **CPU → Interface → I/O → Interface → Memory**

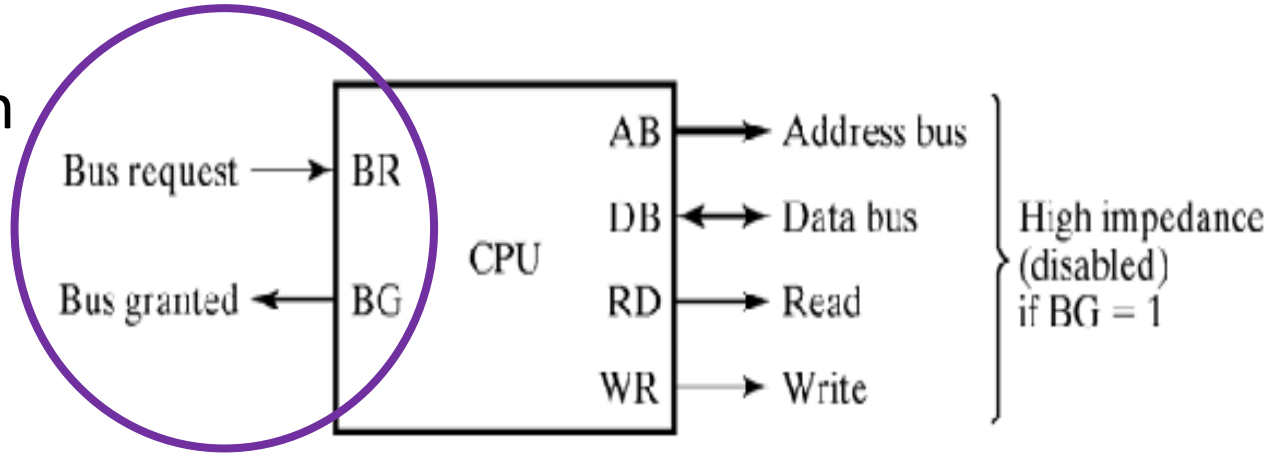
# Direct Memory Access (DMA) - Module Design

## CPU BUS Signals for DMA Transfer

- **BR Signal**
- **BG Signal – issued by CPU**
  - **0** --- DMA communicates with CPU,
  - **1** ---- DMA communicates with Memory

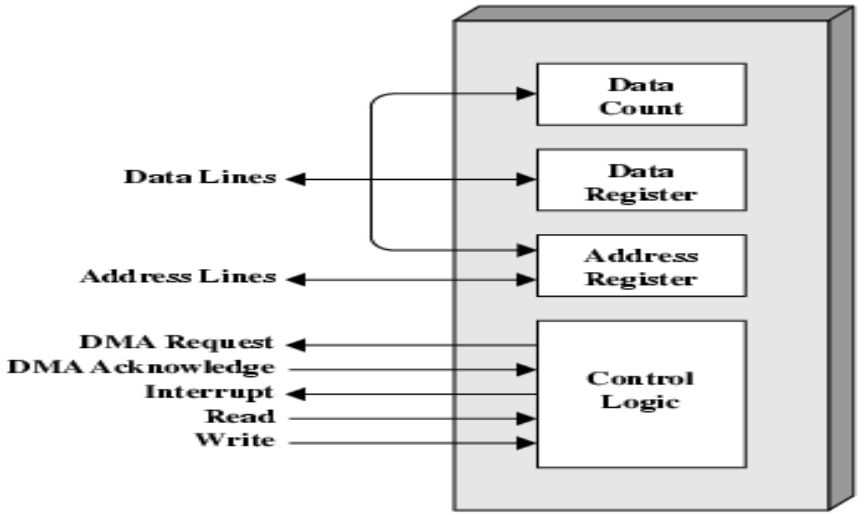
1. DMA Controller sends Bus Request to CPU
2. CPU stops execution of current instruction and places address bus, data bus, read and write lines into high impedance state.
3. CPU activates Bus Grant
4. After transfer, DMA disables BR (Bus Request)
5. CPU continues normal operation

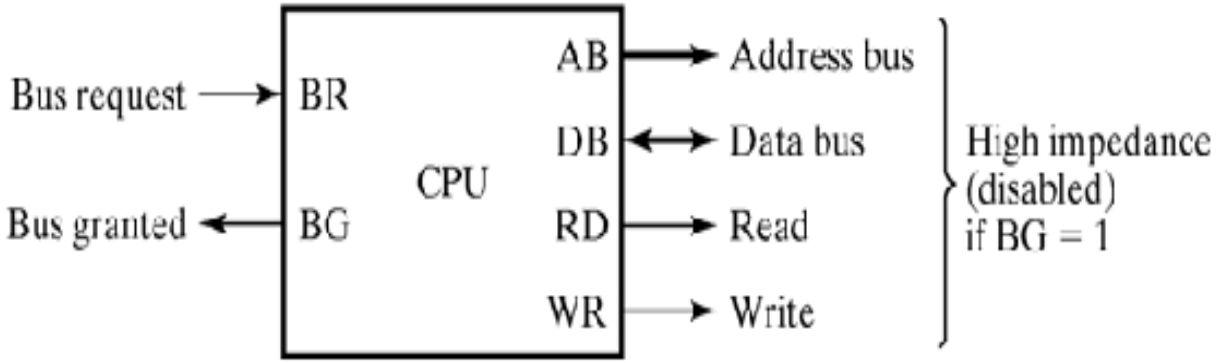


# Direct Memory Access (DMA) - Operation

1. CPU tells DMA controller:-
   - Read/Write
   - Device address
   - Starting address of memory block for data
   - Amount of data to be transferred
2. CPU carries on with other work
3. DMA controller deals with transfer
4. DMA controller sends interrupt when finished



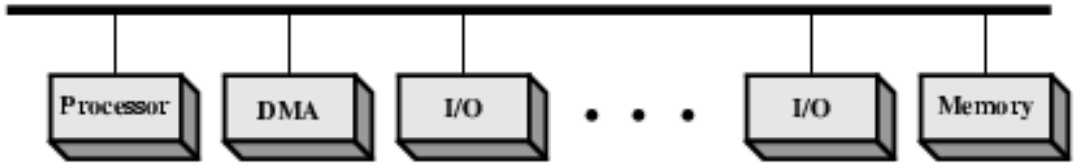



# DMA - Different ways of DMA transfer

- **Burst transfer** – a block sequence consisting of a number of memory words is transferred in a continuous burst.
  - need for fast devices
- **Cycle stealing** – transfer one word at a time, after which it must return control of the buses to the CPU.
  - CPU delays 1 cycle to allow Direct Memory I/O transfer

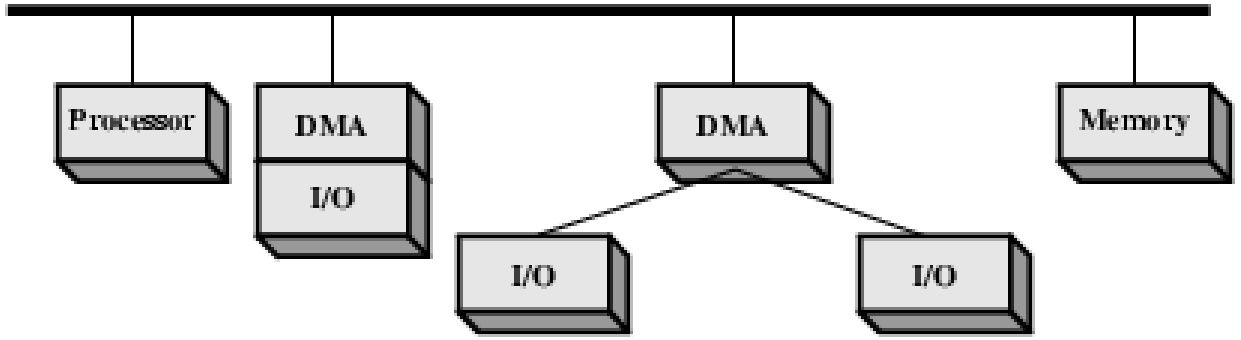
# DMA - Configurations

## Configuration 1: Single Bus, Detached DMA controller



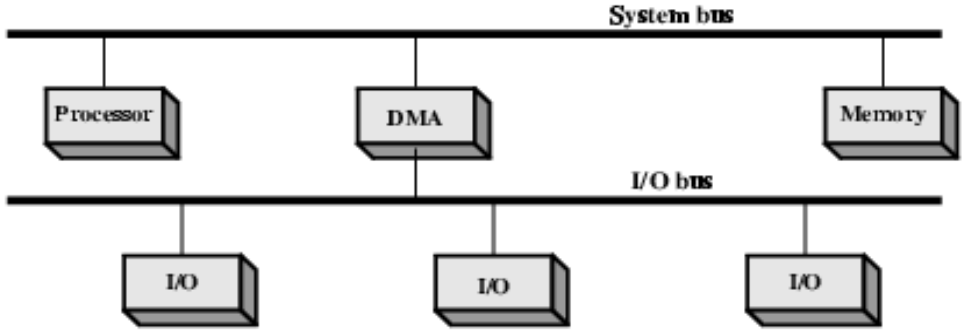

- Each transfer uses bus twice
  - I/O to DMA then DMA to memory
- CPU is suspended twice

## Configuration 2: Single Bus, Integrated DMA controller

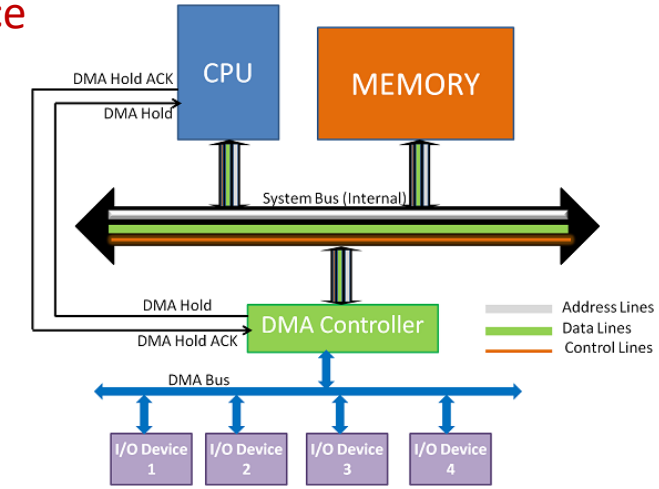



- Controller may support >1 device
- Each transfer uses bus once
  - DMA to memory
- CPU is suspended once

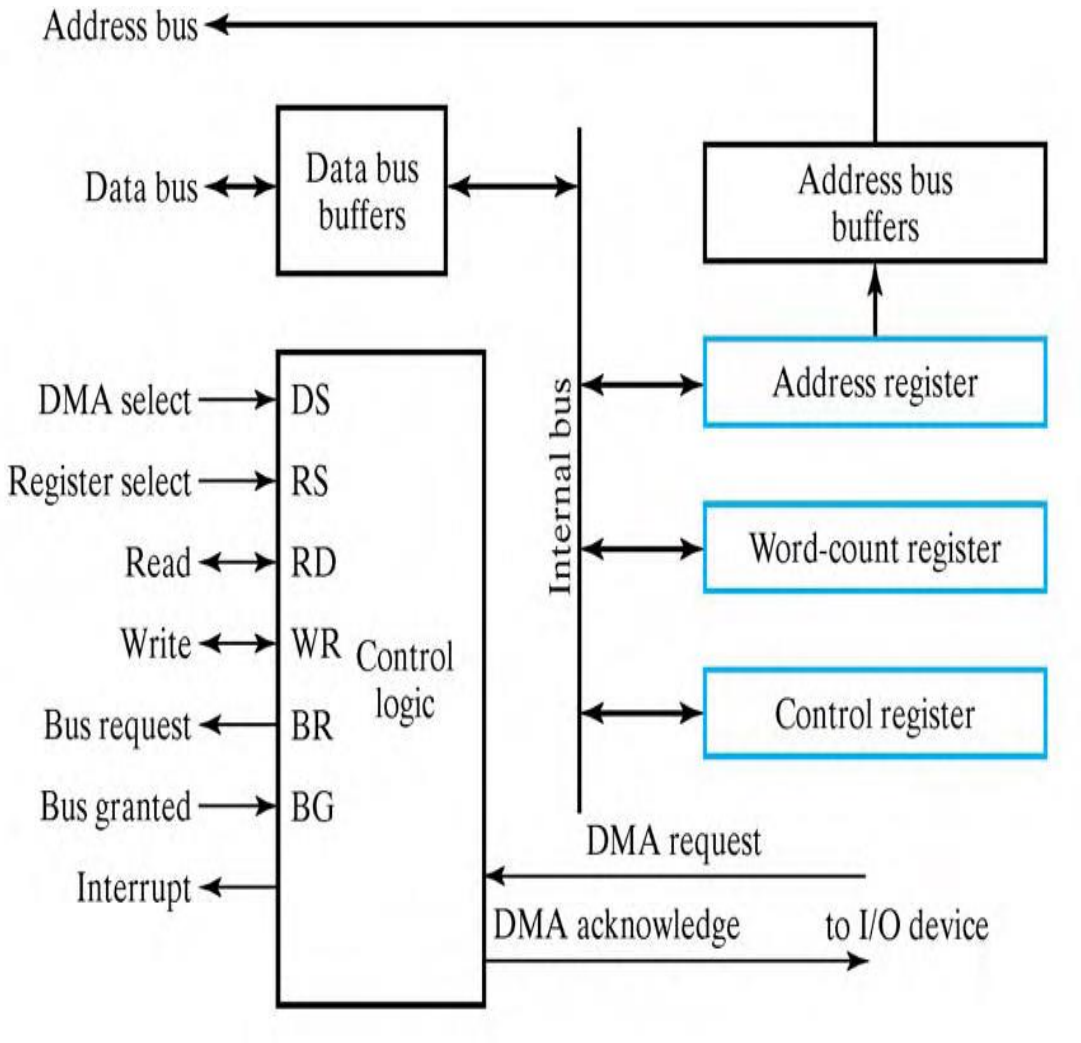
## Configuration 3: Separate I/O Bus




- Bus supports all DMA enabled devices
- Each transfer uses bus once
  - DMA to memory
- CPU is suspended once

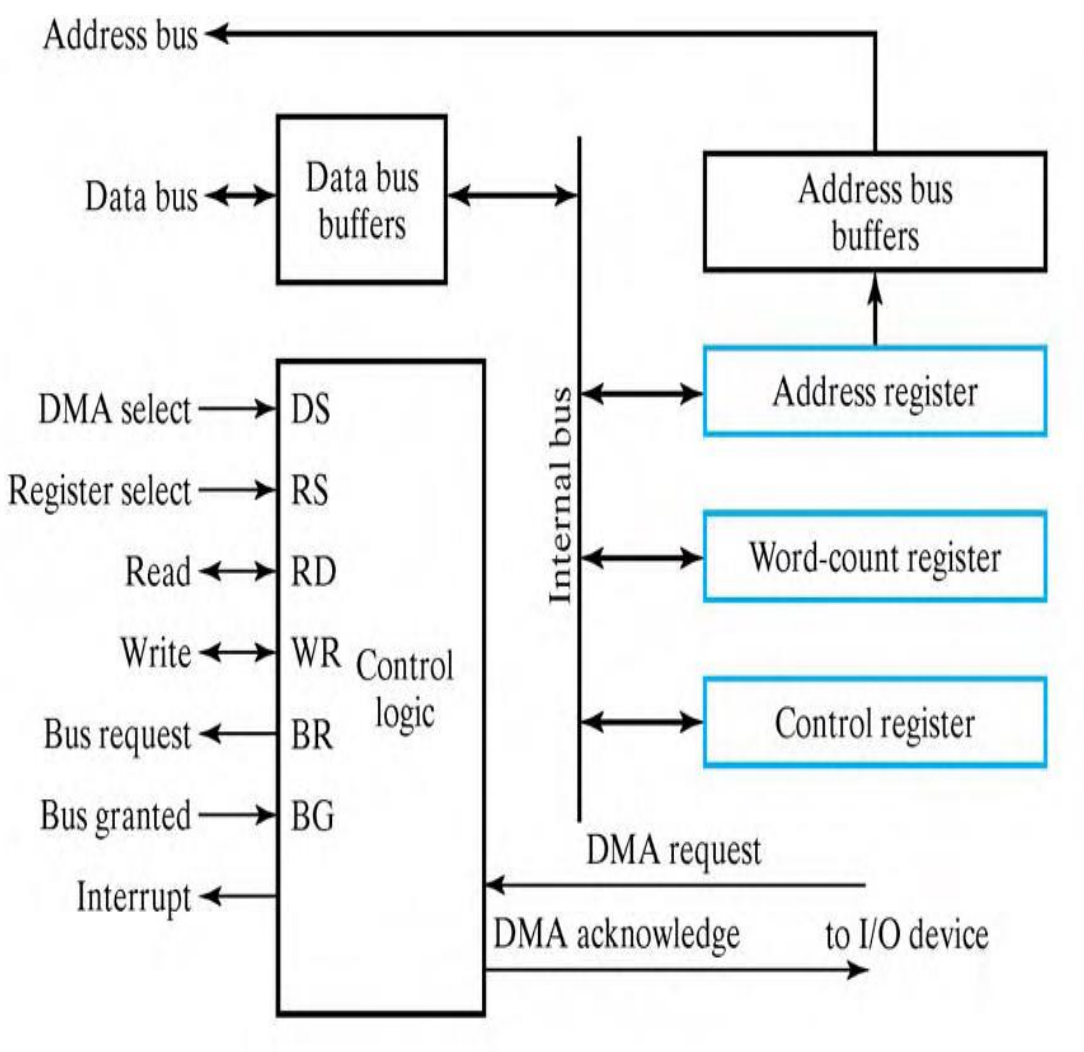


# DMA Controller



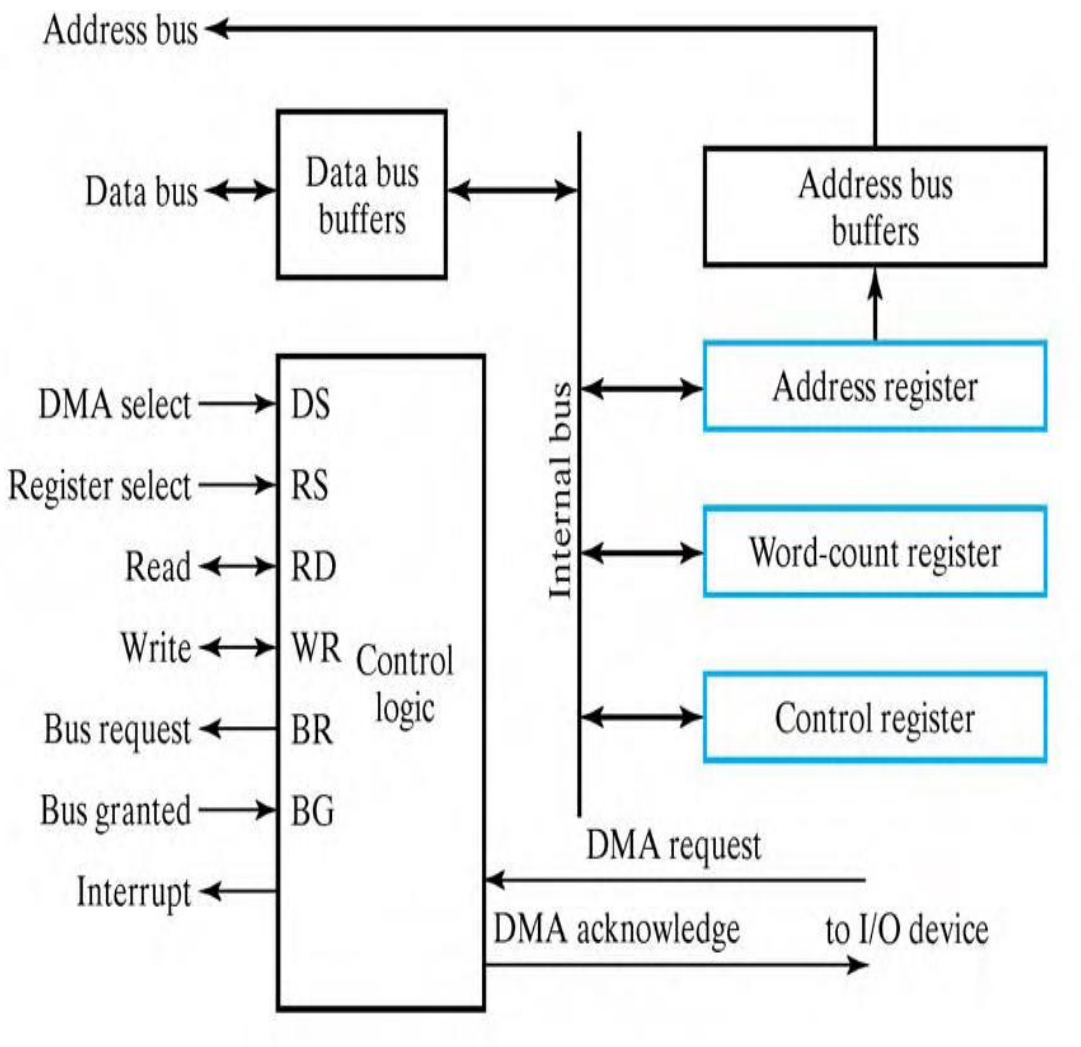

- Three registers
  - Address registers – to specify the desired location in memory
    - Incremented after each word transfer
  - Word count registers – numbers of words to be transferred
    - Decremented after each word transfer and tested for 0.
  - Control register – specifies mode of transfer

# DMA Controller



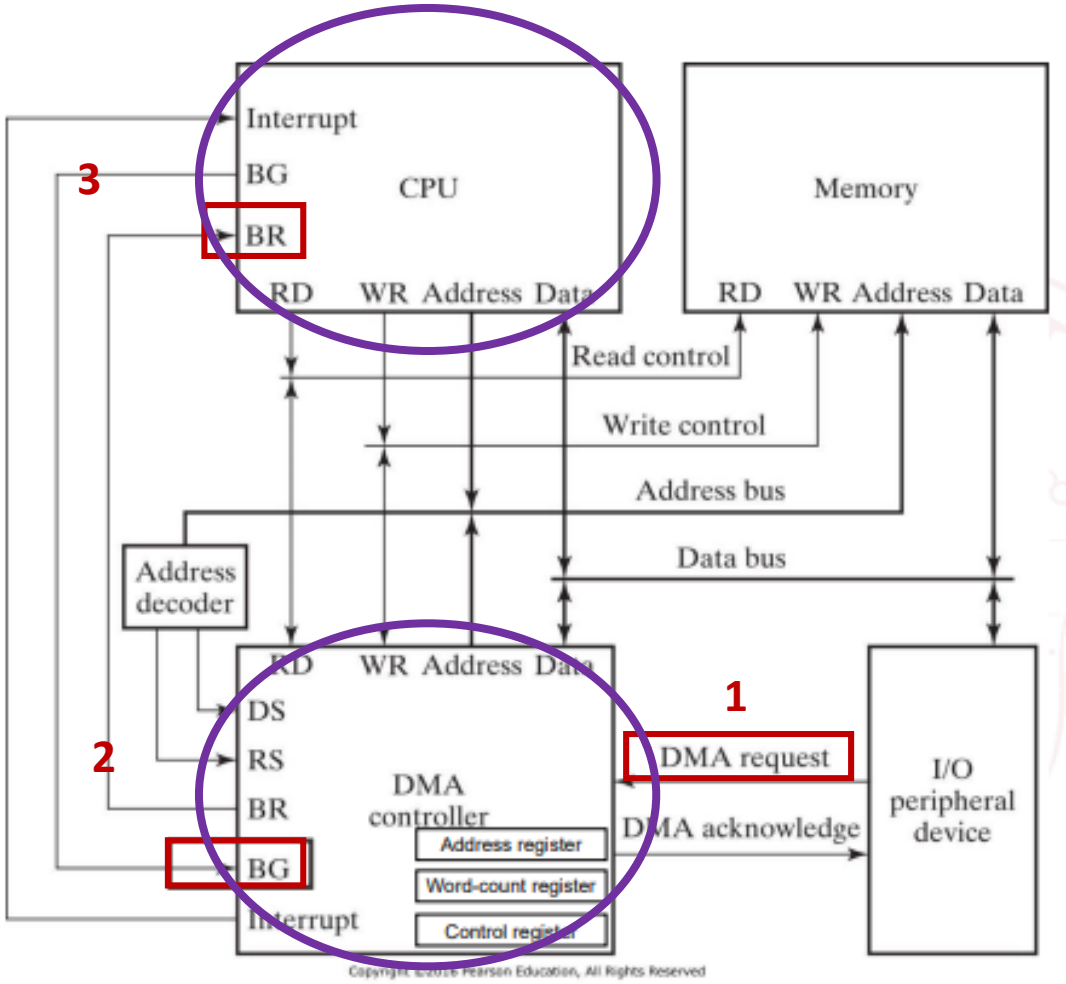

- Data bus and control lines => used to communicate with CPU
- Registers in DMA are selected by CPU through address bus by enabling DS and RS inputs.
- BG = 0 => CPU reads from or writes to the DMA registers
- BG =1 => DMA directly communicates with Memory by specifying address in address bus and activates the RD or WR control.
- DMA Request and acknowledge signals are used to communicate with peripheral devices

# DMA Controller - Initialization



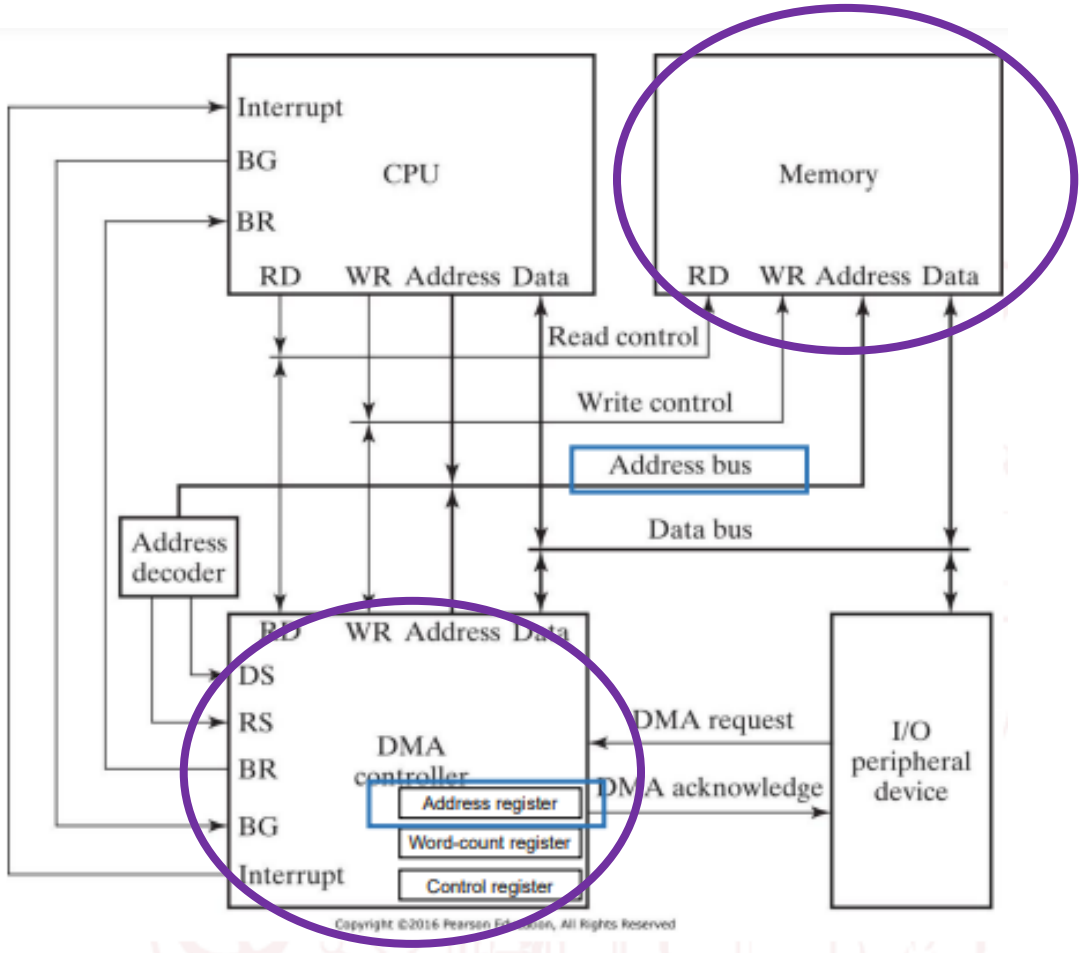

- CPU initializes DMA by sending the following information through data bus
  - Starting address of the memory block where data are available (for read) or where data are to be stored (for write)
  - The word count
  - Control – to read or write
  - A control to start the DMA transfer
- After initialization, CPU stops communicating with DMA unless it receives an interrupt signal or if it wants to checks how many words have been transferred.

# DMA Transfer



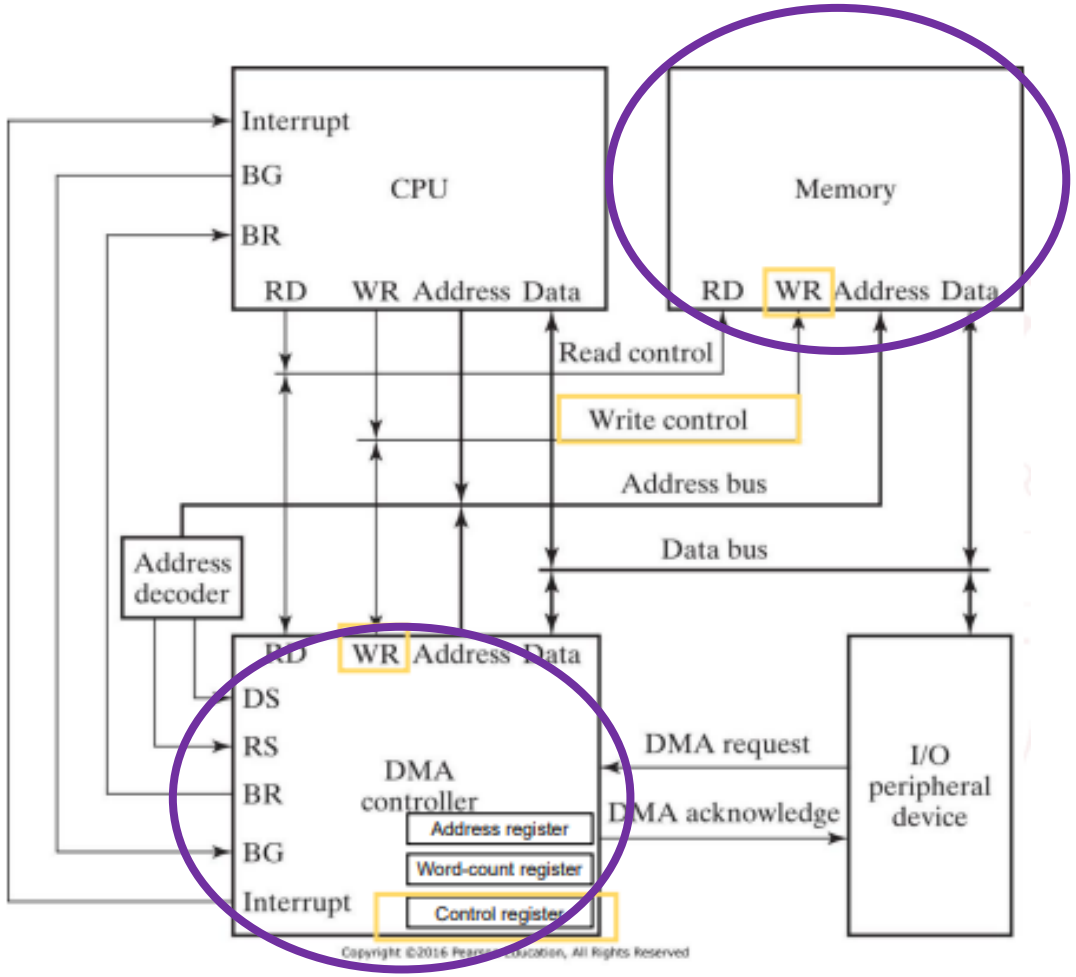

1. Peripheral device **DMA Request** ----------> DMA controller
   **Bus Request** ----------> CPU **Bus grant** ----------> DMA controller

# DMA Transfer



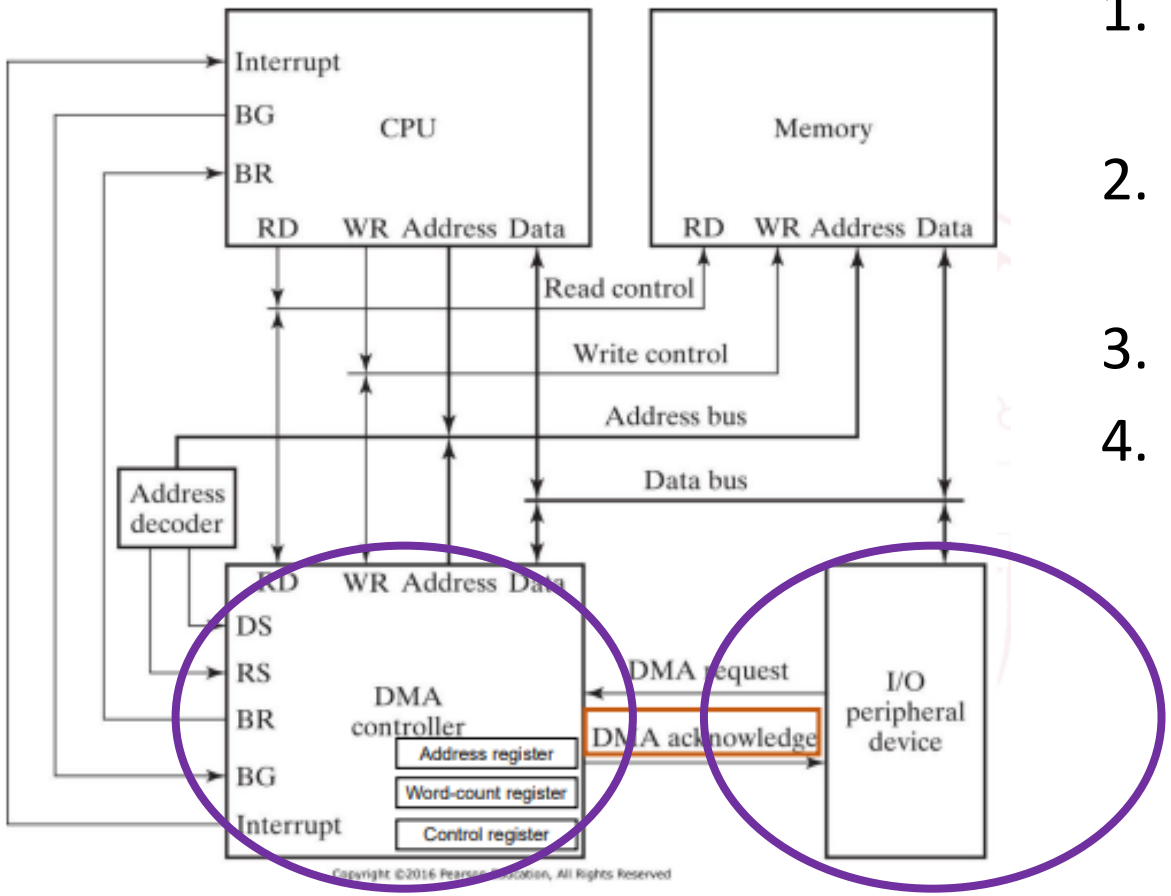

1. Peripheral device ----DMA Request----> DMA controller ----Bus Request----> CPU ----Bus grant----> DMA controller
2. DMA controller puts the current value of its address register onto address bus.

# DMA Transfer



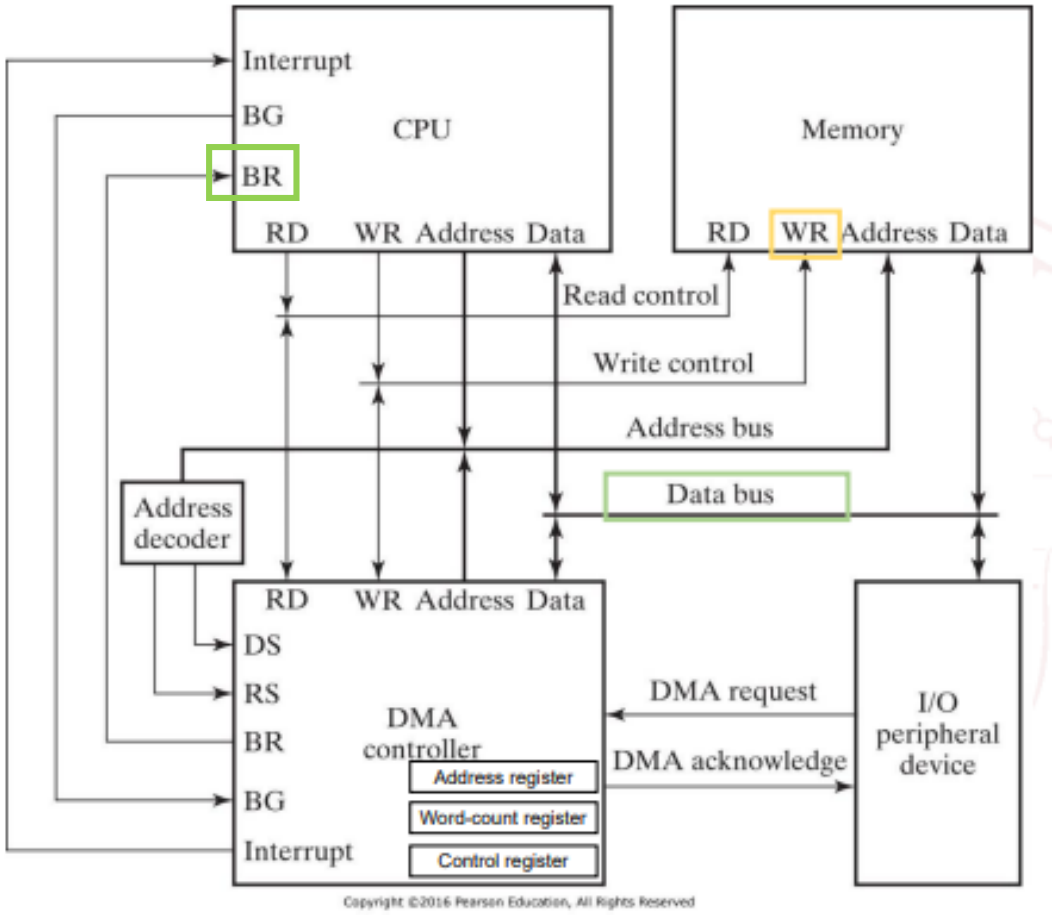

1. Peripheral device **DMA Request** ----------> DMA controller
   **Bus Request** ----------> CPU **Bus grant** ----------> DMA controller
2. DMA controller puts the current value of its address register onto address bus.
3. DMAC activates RD or WR signal.

# DMA Transfer



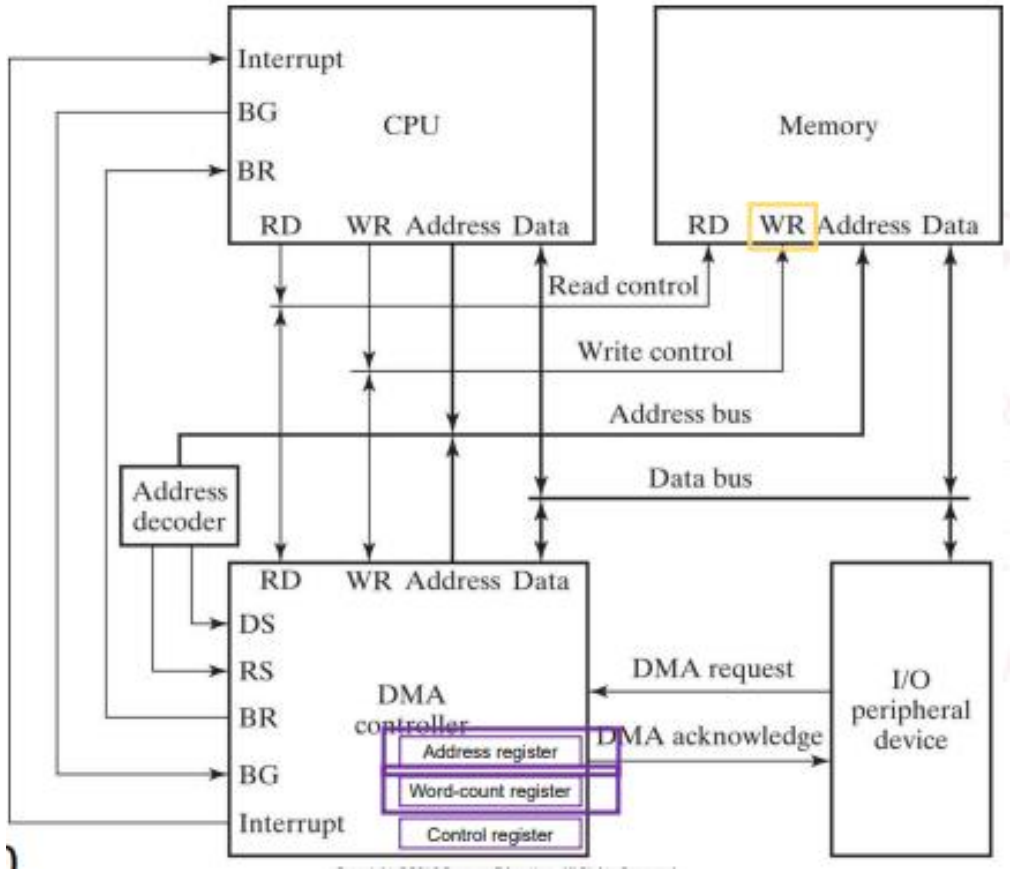

1. Peripheral device ---DMA Request---> DMA controller ---Bus Request---> CPU ---Bus grant---> DMA controller
2. DMA controller puts the current value of its address register onto address bus.
3. DMAC activates RD or WR signal.
4. DMAC ---DMA Acknowledge---> peripheral device

# DMA Transfer




1. Peripheral device ---DMA Request---> DMA controller
   ---Bus Request---> CPU ---Bus grant---> DMA controller
2. DMA controller puts the current value of its address register onto address bus.
3. DMAC activates RD or WR signal.
4. DMAC ---DMA Acknowledge---> peripheral device
5. RD and WR are bidirectional
   1. BR = 0 => CPU communicates with the internal DMA registers
   2. BR = 1 => RD and WR are output lines from DMA controller to the RAM to specify read or write operation for data

# DMA Transfer




1. Peripheral device ---DMA Request---> DMA controller ---Bus Request---> CPU ---Bus grant---> DMA controller
2. DMA controller puts the current value of its address register onto address bus.
3. DMAC activates RD or WR signal.
4. DMAC ---DMA Acknowledge---> peripheral device
5. RD and WR are bidirectional
    1. BR = 0 => CPU communicates with the internal DMA registers
    2. BR = 1 => RD and WR are output lines from DMA controller to the RAM to specify read or write operation for data

# DMA - Application

- Fast transfer of information between magnetic disk and memory
- Updating the display in an interactive terminal

1 Byte = 8 bits
1 KB = 1024 B(ytes) = $2^{10}$ B
1 MB = 1024 KB = $2^{20}$ B
1 GB = 1024 MB = $2^{30}$ B

# DMA - Problems

## Problem 1:

A hard disk with a transfer rate of 20 Mbytes/second is constantly transferring data to memory using DMA. The processor runs at 300MHz, and takes 300 and 900 clock cycles to initiate and complete DMA transfer respectively. If the size of the transfer is 20 Kbytes, what is the percentage of processor time consumed for the transfer operation?

## Solution:

$$Total\ Time = \frac{Total\ Data}{Total\ Transfer\ Rate}$$

$$Processor\ Time = \frac{Cycles\ for\ DMA\ transfer}{Processor\ Speed}$$

$$\%\ Processor\ Time = \frac{Processor\ Time}{Total\ Time} \times 100$$

### Step 1: Find Total Transfer Time

Transfer rate=20 MB per second = $20 \times 2^{20}$ bytes/sec

Data=20 KB=$20 \times 2^{10}$ bytes

Time=$(20 \times 2^{10})/(20 \times 2^{20}) = 1 \times 2^{-10} = 1 \times 10^{-3}$ =1 ms

### Step 2 :Find Processor Time

**1 MHz = $10^6$ cycles/sec**

Processor speed= 300 MHz = $300 \times 10^6$ Cycles/sec

Cycles required by CPU for DMA Transfer =300+900 =1200 cycles

Time=1200 cycles/$(300 \times 10^6)$ cycles/sec = $4 \times 10^{-6}$ sec = .004 ms

### Step 3 : Find % of Processor Time

% Processor Time = (.004/1) × 100 = 0.4%

| Term | Normal Usage | Usage as Power of 2 |
|---|---|---|
| K ( Kilo) | $10^3$ | $2^{10}$ = 1,024 |
| M (Mega) | $10^6$ | $2^{20}$ = 1,048,576 |
| G (Giga) | $10^9$ | $2^{30}$ = 1,073,741,824 |
| T (Tera) | $10^{12}$ | $2^{40}$ = 1,099,511,627,776 |

# DMA - Problems

| | | | |
|---|---|---|---|
| 1 Byte | = | 8 bits | |
| 1 KB | = | 1024 B(ytes) | = $2^{10}$ B |
| 1 MB | = | 1024 KB | = $2^{20}$ B |
| 1 GB | = | 1024 MB | = $2^{30}$ B |

## Problem 2:

A hard disk with a transfer rate of 10 Mbytes/second is constantly transferring data to memory using DMA. The processor runs at 600MHz, and takes 300 and 900 clock cycles to initiate and complete DMA transfer respectively. If the size of the transfer is 20 Kbytes, what is the percentage of processor time consumed for the transfer operation?

## Solution:

$$Total\ Time = \frac{Total\ Data}{Total\ Transfer\ Rate}$$

### Step 1: Find Total Transfer Time

Transfer rate=10 MB per second = $10 \times 2^{20}$ bytes/sec

Data=20 KB=$20 \times 2^{10}$ bytes

Time=$(20 \times 2^{10})/(10 \times 2^{20}) = 2 \times 2^{-10} = 2 \times 10^{-3} = 2$ ms

$$Processor\ Time = \frac{Cycles\ for\ DMA\ transfer}{Processor\ Speed}$$

$$\%\ Processor\ Time = \frac{Processor\ Time}{Total\ Time} \times 100$$

### Step 2 :Find Processor Time

**1 MHz = $10^6$ cycles/sec**

Processor speed= 600 MHz = $600 \times 10^6$ Cycles/sec

Cycles required by CPU for DMA Transfer =300+900 =1200 cycles

Time=1200 cycles/$(600 \times 10^6)$ cycles/sec = $2 \times 10^{-6}$ sec = .002 ms

### Step 3 : Find % of Processor Time

% Processor Time = $(.002/2) \times 100 = 0.1\%$

| Term | Normal Usage | Usage as Power of 2 |
|---|---|---|
| K ( Kilo) | $10^3$ | $2^{10} = 1,024$ |
| M (Mega) | $10^6$ | $2^{20} = 1,048,576$ |
| G (Giga) | $10^9$ | $2^{30} = 1,073,741,824$ |
| T (Tera) | $10^{12}$ | $2^{40} = 1,099,511,627,776$ |

# DMA - Problems

| | | | |
|---|---|---|---|
| 1 Byte | = | 8 bits | |
| 1 KB | = | 1024 B(ytes) | = $2^{10}$ B |
| 1 MB | = | 1024 KB | = $2^{20}$ B |
| 1 GB | = | 1024 MB | = $2^{30}$ B |

## Problem 3:

1. The size of the data count register of a DMA controller is 16 bits. The processor needs to transfer a file of 29,154 kilobytes from disk to main memory. The memory is byte addressable. The minimum number of times the DMA controller needs to get the control of the system bus from the processor to transfer the file from the disk to main memory is_____

Data count register = 16 bits.

=> Data that can be transferred at once = 2^16 bytes

(Because byte addressable memory)

Total data to be transferred = 29154∗1024 bytes.

Number of transfers required

= 29154∗1024÷65536=455.531

So, **456** attempts required.

Data count register provides the number of words DMA can transfer in one cycle.

If memory is byte addressable; 1 word = 1 byte

1 kilobytes = $2^{10}$ bytes

**Calculation:**

Size of data counter register of DMA controller = 16 bits

It means, $2^{16}$ words can be transferred in one cycle.

As, memory is byte addressable.

So, $2^{16}$ bytes in one cycle. ($2^{16}$ bytes = $2^{6}$ kilobytes)

File size = 29154 kilobytes

Minimum number of times the DMA controller needs to get control of the system bus from the processor to transfer the file from the disk to main memory is =

$$\left\lceil \frac{File\ size}{bytes\ in\ one\ cycle} \right\rceil = \left\lceil \frac{29154\ KB}{2^{16}} \right\rceil = 456$$

# DMA - Problems

| | | | |
|---|---|---|---|
| 1 Byte | = | 8 bits | |
| 1 KB | = | 1024 B(ytes) | = $2^{10}$ B |
| 1 MB | = | 1024 KB | = $2^{20}$ B |
| 1 GB | = | 1024 MB | = $2^{30}$ B |

**Problem 4:**

- The processor needs to transfer a file of 29154 kilobytes from disk to main memory. The memory is byte addressable. The size of data count register of a DMA controller is 16 bits. Calculate the minimum number of times the DMA controller needs to get the control of system bus from the processor to transfer the complete file from disk to main memory in following modes?
  - i. Cycle stealing mode
  - ii. Block transfer/Burst transfer mode

# DMA - Problems

| | | | |
|---|---|---|---|
| 1 Byte | = | 8 bits | |
| 1 KB | = | 1024 B(ytes) | = $2^{10}$ B |
| 1 MB | = | 1024 KB | = $2^{20}$ B |
| 1 GB | = | 1024 MB | = $2^{30}$ B |

## Problem 4: Solution

> *"Data count register gives the number of words the DMA can transfer in a single cycle..*

Here it is $16$ bits.. so max $2^{16}$ words can be transferred in one cycle..

Since memory is byte addressable.. $1\ \text{word} = 1\ \text{byte}$

so $2^{16}$ bytes in $1$ cycle..

Now for the given file..

$$\text{File size} = 29154\ \text{KB} = 29154 \times 2^{10}\ \text{B}$$

$$1\ \text{cylce} \rightarrow \text{DMA transfers}\ 2^{16}\ \text{B}$$

**i.e**

$$1\ B\ \text{transfered by DMA} \rightarrow \frac{1}{2^{16}}\ \text{cycles.}$$

Now, for full file of size $29154$ KB,

$$\text{Minimum number of cylces} = \frac{(29154 \times 2^{10}\ B)}{2^{16}} = 455.53$$

But number of cylces is asked so $455.53 \rightarrow 456.$

# DMA - Problems

**For more Problems on DMA**

https://gateoverflow.in/tag/dma

# Direct Cache Access

- DCA - a system protocol that allows an input/output (I/O) device to load data directly into a cache.
- This cache is then used by the network stack process for immediate access.
- DCA reduces the number of memory access operations needed to process each packet, which increases throughput and reduces CPU load.
- DCA also eliminates the processor cycles required to read packet headers and descriptors from system memory.

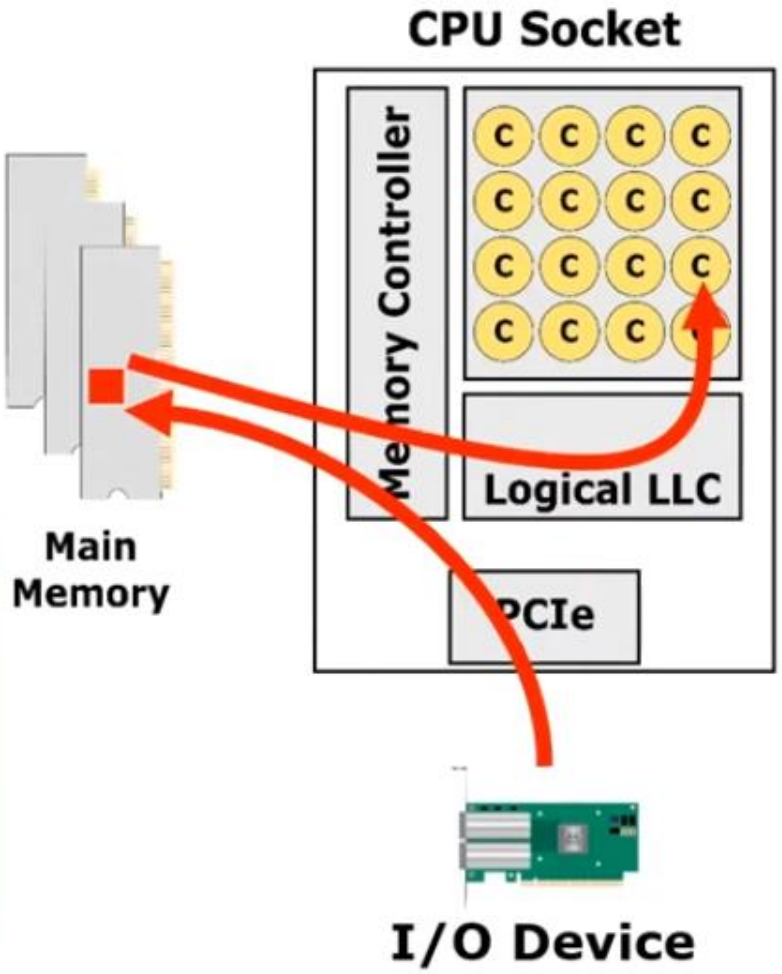
# Direct Cache Access

- DCA improves DMA by changing its target from memory to the processor cache.
- DMA frees the processor from heavy memory copy in I/O operations, which increases processor utilization when dealing with high-speed I/O devices.

# Direct Cache Access

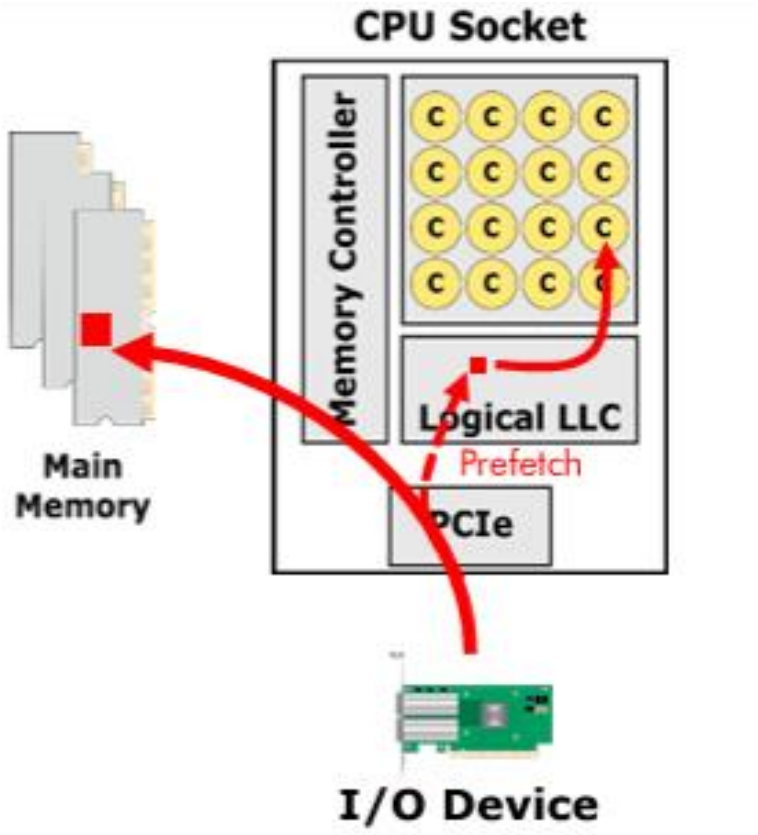
1. I/O device DMAs* packets to main memory
2. CPU later fetches them to cache

Inefficient:
- Large number of accesses to main memory
- High access latency (>60ns)
- Unnecessary memory bandwidth usage



# Direct Cache Access

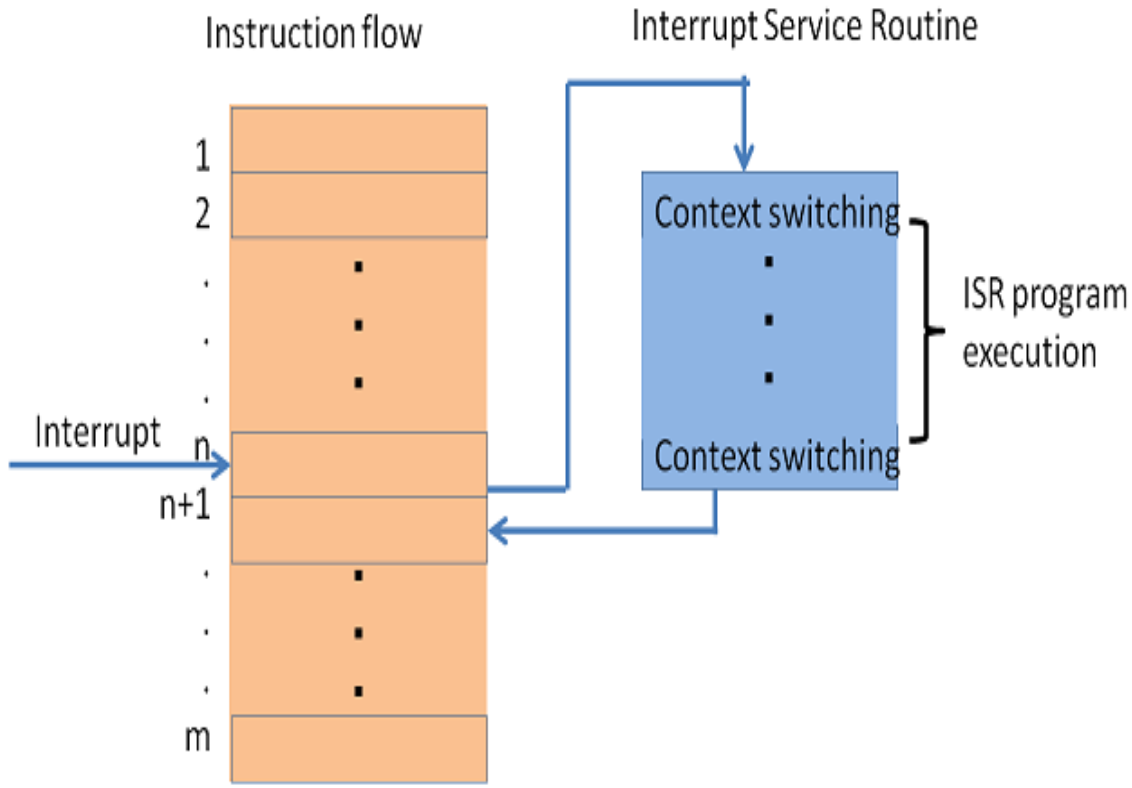
- Direct Cache Access enables a capable I/O device such as a network controller to activate a pre-fetch engine in the CPU that loads arriving packet data into the CPU cache for immediate access by the network stack process.
- CPU later fetches them from cache
- Prefetching operation - often based on **hints** provided by Transaction Layer Packet (TLP) processing hints (TPH)
- These hints may indicate which data blocks are likely to be accessed soon, allowing the CPU to prefetch them into the cache for faster access.



# Interrupt Structures

Interrupt:

- An interrupt is a signal from a device attached to a computer or from a program within the computer that causes the CPU to stop its normal program execution and perform service related to the event.



# Vectored Interrupt

- In vectored interrupts, a device requesting an interrupt identifies itself directly by sending a special code to the processor over the bus.
- This enables the processor to identify the device that generated the interrupt.
- The special code can be the starting address of the ISR or where the ISR is located in memory and is called the interrupt vector.
- when an interrupt occurs, the interrupting device or component provides the processor with additional information called a vector.
- This vector is essentially a numeric identifier or address that points to the specific interrupt service routine (ISR) associated with the interrupting device.

# Vectored Interrupt

**Steps:**

- Interrupt Occurs: Some event or condition triggers an interrupt.
- Vector Provided: The interrupting device sends a vector to the processor. This vector is a unique identifier that helps the processor determine the specific service routine to execute.
- Interrupt Service Routine (ISR): The processor uses the vector to locate the appropriate ISR in a predefined table or memory location. The ISR is a piece of code that handles the specific interrupt.
- Processing: The processor jumps to the address specified by the vector and starts executing the ISR. The ISR performs the necessary actions to handle the interrupt, which may include saving the current state of the processor, performing specific tasks related to the interrupt, and restoring the processor's state afterward.
- Return to Normal Execution: After the ISR completes its tasks, the processor returns to the interrupted program or task.

# Prioritized Interrupt

- The I/O devices are organized in a priority structure such that the **interrupt raised by the high priority device is accepted** even if the processor servicing the interrupt from a low priority device
- A priority level is assigned to the processor which can be regulated using the program Now, whenever a processor starts the execution of some program its priority level is set equal to the priority of the program in execution.
- Thus while executing the current program the processor only accepts the interrupts from **the device that has higher priority as of the processor**

# Prioritized Interrupt

- Now, when the processor is executing an interrupt service routine the processor priority level is set to the priority of the device of which the interrupt processor is servicing
- Thus the processor only accepts the interrupts from the device with the higher priority and ignore the interrupts from the device with the same or low priority
- To set the priority level of the processor some bits of the processor's status register is used

# Prioritized Interrupt

Maskable Interrupt

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

Non Maskable Interrupt

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

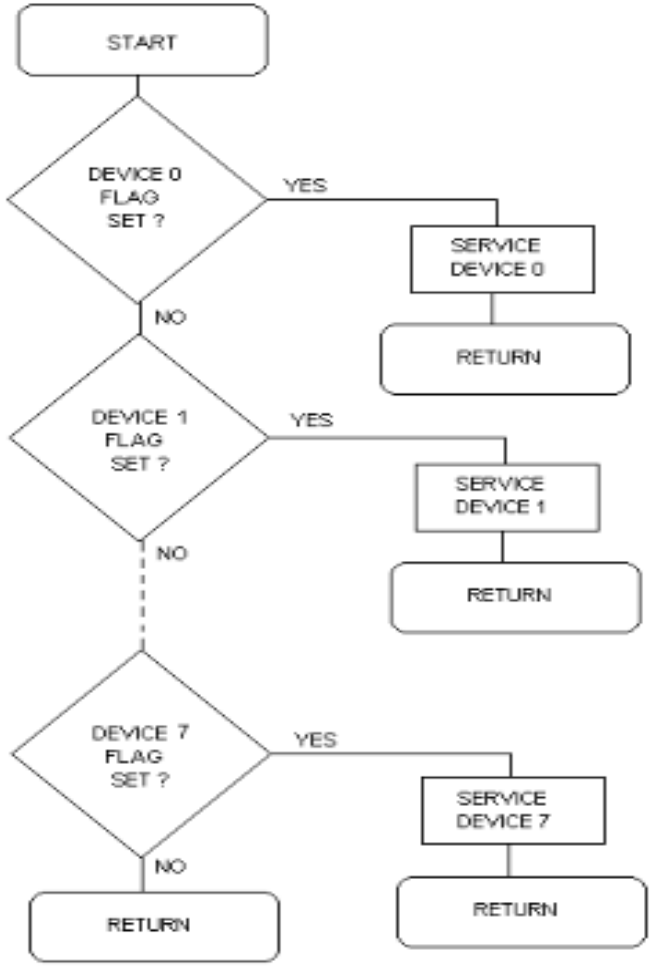
# Prioritized Interrupt

- When two devices interrupt the computer at the same time, the computer services the devices with the higher priority first.
- Establishing the priority of simultaneous interrupts can be done by **software or hardware:**
  - **Software:** Polling
  - **Hardware:** Daisy chain, Parallel Priority

# Prioritized Interrupt – Software Method

## 1.Polling

- A polling procedure is used to identify the highest-priority source by software means.
- In this method there is one common branch address for all interrupts.
- The interrupt handling program begins at the branch address and polls the interrupt sources in sequence.
- The order in which they are tested determines the priority of each interrupt.
- The highest-priority source is tested first, and if its interrupt signal is on, control branches to a service routine for this source.
- Otherwise, the next-lower-priority source is tested, and so on




SOFTWARE POLLING FLOWCHART

# Prioritized Interrupt – Software Method

## 1.Polling(Software Method)

- In this method, all interrupts are serviced by branching to the same service program. There is one common branch address for all the interrupts
- This program then checks with each device if it is the one generating the interrupt.
- The order of checking is determined by the priority that has to be set. The device having the highest priority is checked first and then devices are checked in descending order of priority.

# Prioritized Interrupt – Hardware Method

- It accepts interrupt requests from many sources, determines which of the incoming requests has the highest priority, and issues an interrupt request to the computer based on this determination.
- To speed up the operation, each interrupt source has its own interrupt vector to access its own service routine directly.
- The hardware priority function can be established by either a serial or a parallel connection of interrupt lines.
- The serial connection is also known as the **daisy chaining** method

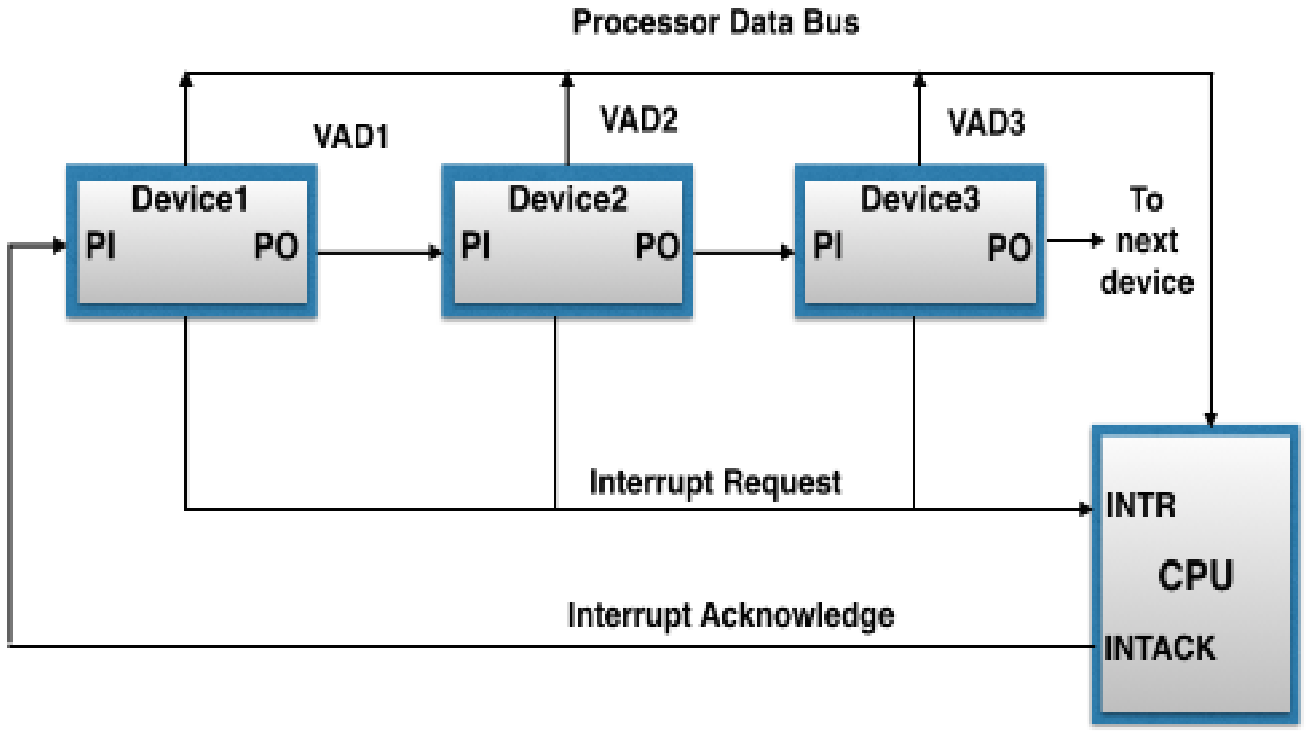
# Prioritized Interrupt – Hardware Method

## 2.Daisy Chaining method

- The daisy-chaining method involves connecting all the devices that can request an interrupt in a serial manner.
- This configuration is governed by the priority of the devices. The device with the highest priority is placed first followed by the second highest priority device and so on.

# Prioritized Interrupt – Hardware Method

## 2.Daisy Chaining method(Hardware method)




•When no interrupts are pending, the line is in HIGH state

•. But if any of the devices raises an interrupt, it places the interrupt request line in the LOW state.

•CPU acknowledges this interrupt request and signal is received at the PI(Priority in) input of device 1.

•If device has not requested the interrupt,

**(PI = 1 & PO = 1)**

- if the device had requested the interrupt, (**PI =1 & PO = 0**)
- device consumes the acknowledge signal and block its further use by placing 0 at its PO(priority out) output
- device then proceeds to place its interrupt vector address(VAD)
- If a device gets 0 at its PI input, it generates 0 at the PO output to tell other devices that acknowledge signal has been blocked**. (PI = 0 & PO = 0)**

# Prioritized Interrupt – Hardware Method

•When no interrupts are pending, the line is in HIGH state. But if any of the devices raises an interrupt, it places the interrupt request line in the LOW state.

•The CPU acknowledges this interrupt request from the line and then enables the interrupt acknowledge line in response to the request.

•This signal is received at the PI(Priority in) input of device 1.

•If the device has not requested the interrupt, it passes this signal to the next device through its PO(priority out) output. **(PI = 1 & PO = 1)**

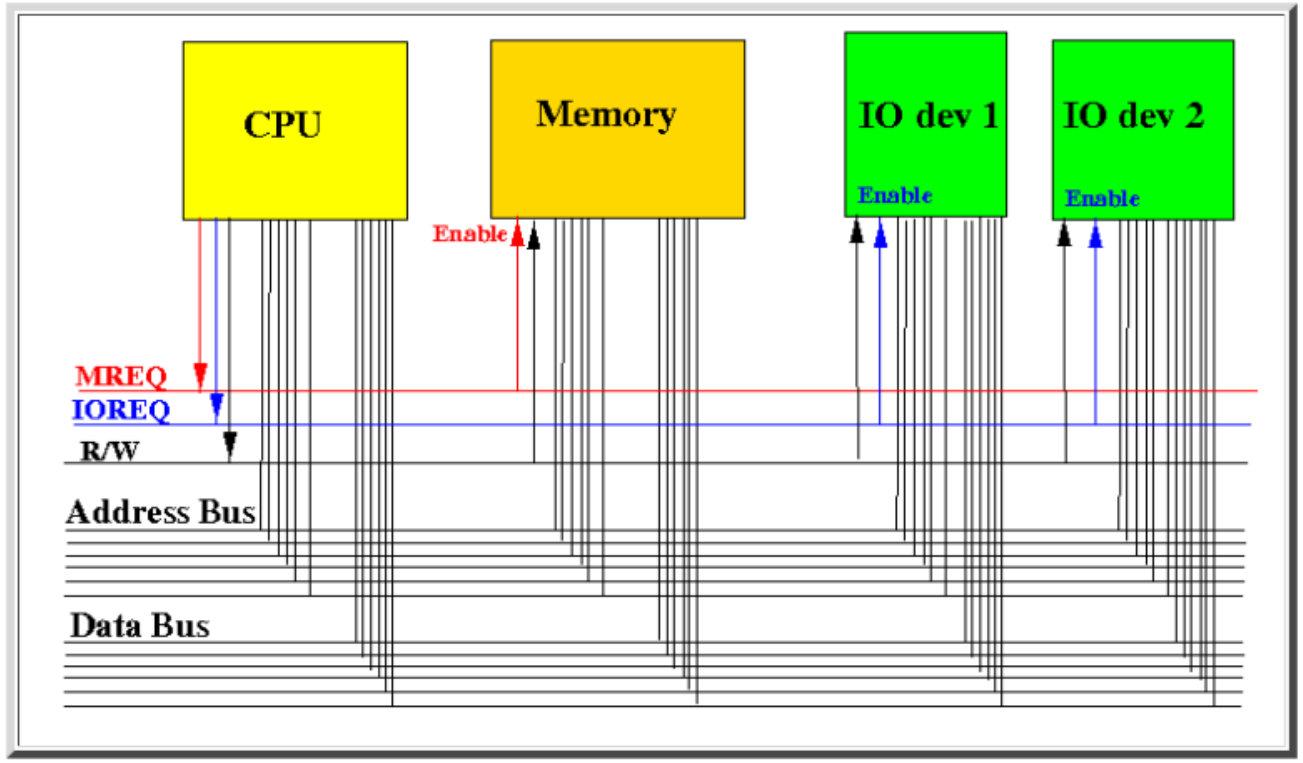
•However, if the device had requested the interrupt, (**PI =1 & PO = 0**)

- The device consumes the acknowledge signal and block its further use by placing 0 at its PO(priority out) output.
- The device then proceeds to place its interrupt vector address(VAD) into the data bus of CPU.
- The device puts its interrupt request signal in HIGH state to indicate its interrupt has been taken care of.

•If a **device gets 0 at its PI input, it generates 0 at the PO output to tell other devices that acknowledge signal has been blocked. (PI = 0 & PO = 0)**

# Buses

- “Bus" refers to a communication system that enables the transfer of data, addresses, and control signals between various components of a computer system.
- It serves as a shared pathway or channel through which information flows between different hardware components.



# Synchronous Bus

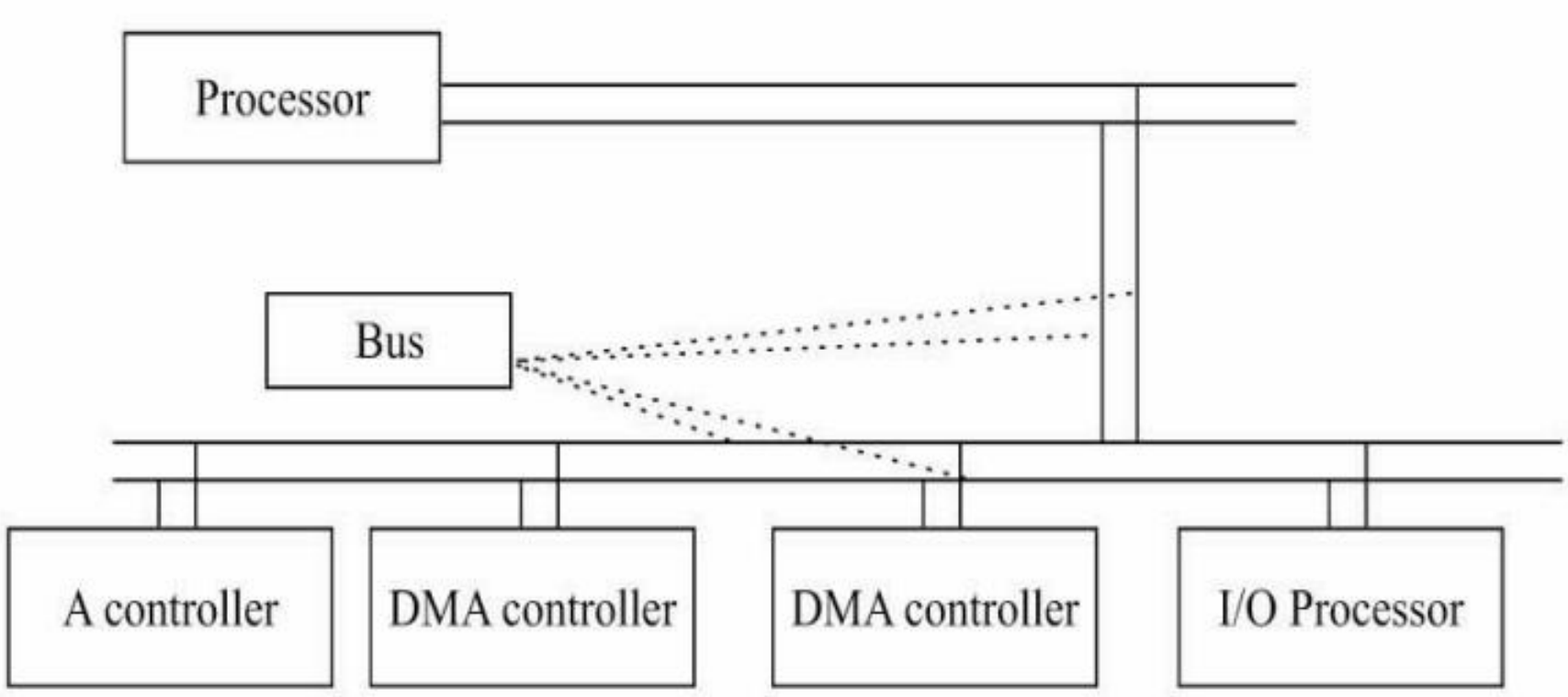
- Synchronous bus (e.g., processor-memory buses)
- Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
- Advantage: involves very little logic and can run very fast
- Disadvantages:
- Every device communicating on the bus must use same clock rate

# Asynchronous Bus

- Asynchronous bus (e.g., I/O buses)
- It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq,Ack, DataRdy)
- Advantages: Can accommodate a wide range of devices and device speeds
- Disadvantage: slow(er)

# Bus Arbitration

- Arbitration - Acquiring the control over bus
- A bus (any bus) cannot be used by more than one device at one time due to electrical properties of the devices.



# Bus Arbitration

- Before any device is allowed to perform a read/write operation using the system bus, it must first obtain permission.
- Master – Slave
- A device that starts a read/write operation is called a **master device**
- The device that it "talks" to is called the **slave device**
- Only **one processor or controller can be bus master**
- The bus master– the controller that has access to a bus at an instance.
- Any one controller or processor can be the bus master at the given instance (s).

# Bus Arbitration

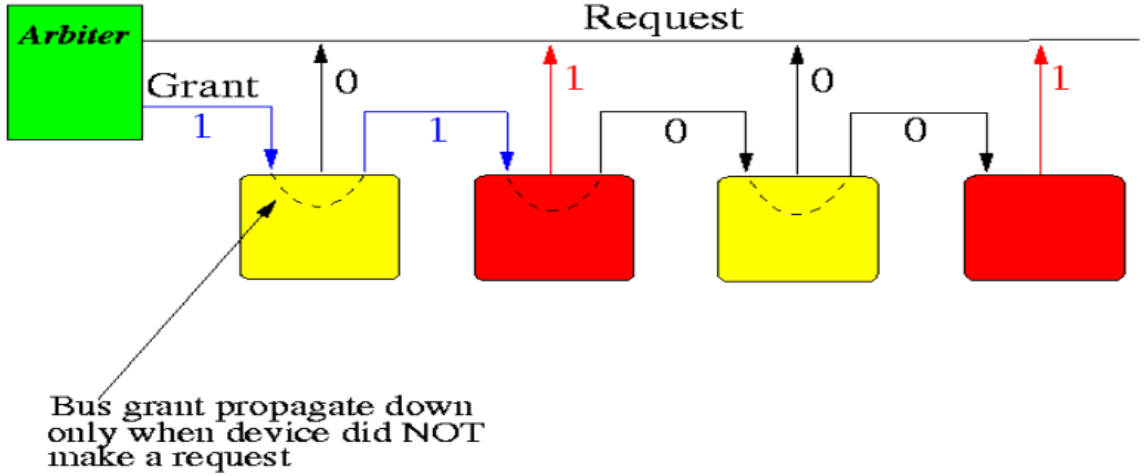
## BUS ARBITRATION METHODS

1. Daisy Chain
2. Independent Bus Requests and Grant
3. Polling

# Bus Arbitration

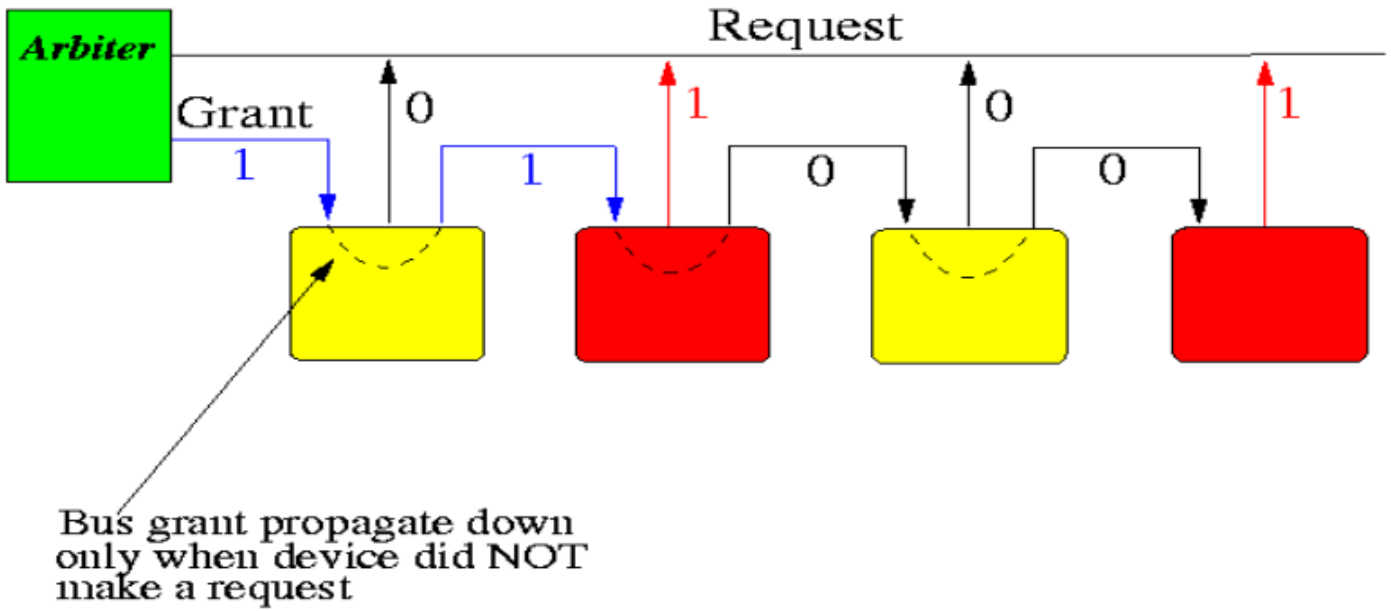
## 1. Daisy Chain

- It is a simple and cheaper method where **all the bus masters use the same line for making bus requests.**
- The **bus grant signal serially propagates through each master** until it encounters the first one that is requesting access to the bus.
- This **master blocks the propagation of the bus grant signal**, therefore any other requesting module will not receive the grant signal and hence cannot access the bus.
- During any bus cycle, the bus master may be any device – the processor or any DMA controller unit, connected to the bus.



# Bus Arbitration

## 1. Daisy Chain




**Advantages:**

- Simplicity and Scalability.
- The user can add more devices anywhere along the chain, up to a certain maximum value.

**Disadvantages:**

•The value of priority assigned to a device depends on the position of the master bus.

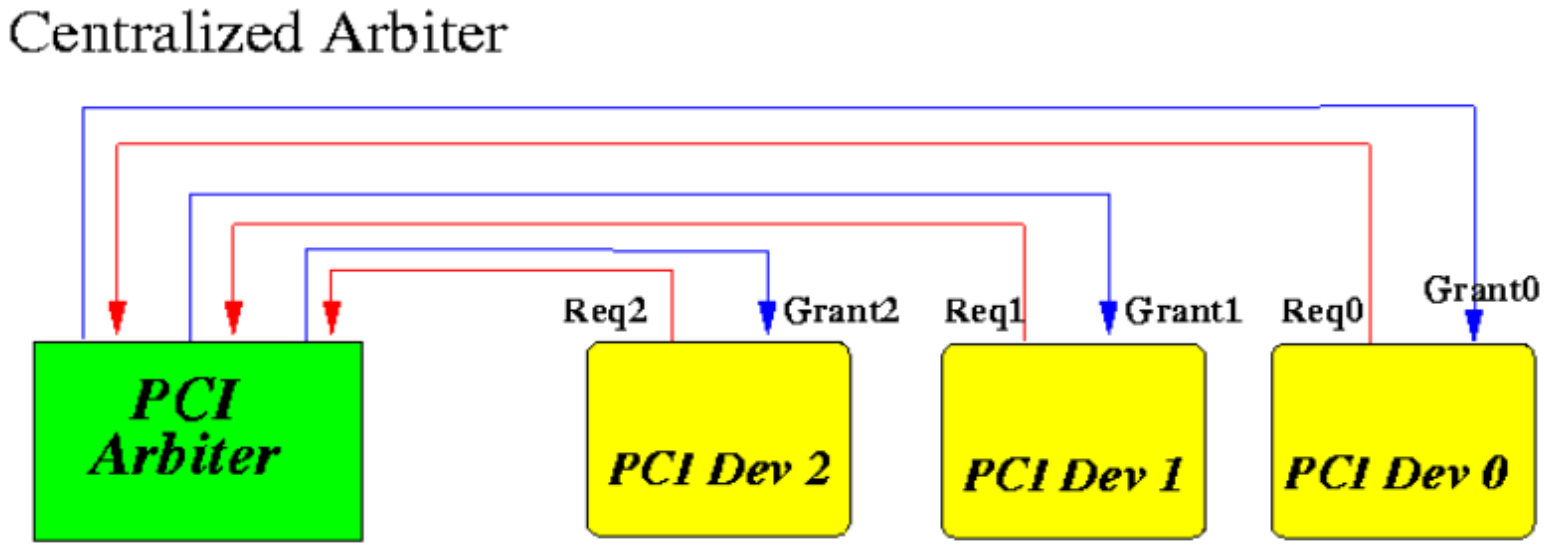
•Propagation delay arises in this method.

•If one device fails then the entire system will stop working.

# Bus Arbitration

## 2. Independent Bus Requests and Grant

- In this, each master has a separate pair of bus request and bus grant lines and **each pair has a priority assigned to it**.
- The built-in priority decoder within the controller selects the highest priority request and asserts the corresponding bus grant signal.



# Bus Arbitration

## 2. Independent Bus Requests and Grant

- Advantages

  This method generates a fast response.

- Disadvantages

  Hardware cost is high as a large no. of control lines is required.

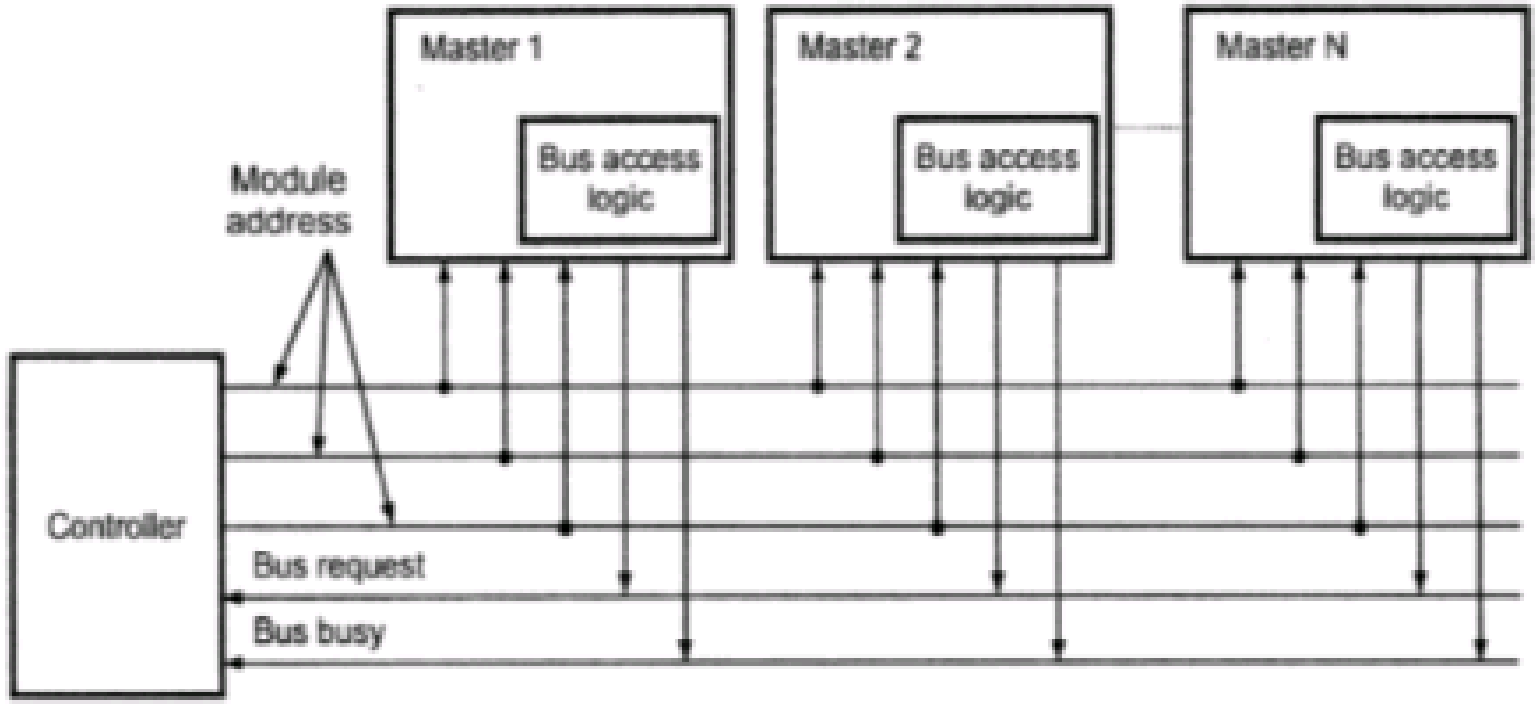
# Bus Arbitration

## 3. Polling

- In this, the **controller is used to generate the address for the master(unique priority),** the number of address lines required depends on the number of masters connected in the system.
- The controller generates a sequence of master addresses. When the **requesting master recognizes its address**, it activates the busy line and begins to use the bus.

# Bus Arbitration

## 3. Polling




- In this the controller is used to generate the addresses for the master.
- Number of address line required depends on the number of master connected in the system.
- For example, if there are 8 masters connected in the system, at least three address lines are required.

# Bus Arbitration

## 3. Polling

**Advantages**

- This method does not favor any particular device and processor.
- The method is also quite simple.
- If one device fails then the entire system will not stop working.

**Disadvantages**

- Adding bus masters is difficult as increases the number of address lines of the circuit.